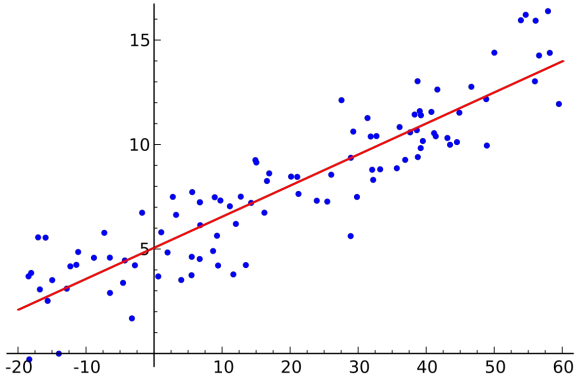
Deep Learning Basic

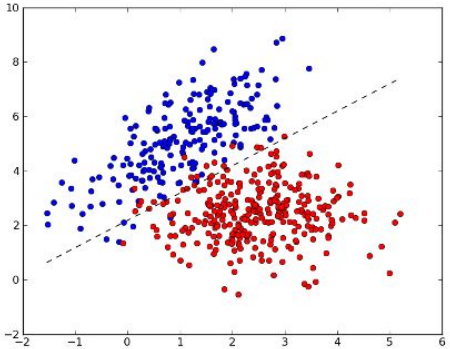
李智揚

FUSION

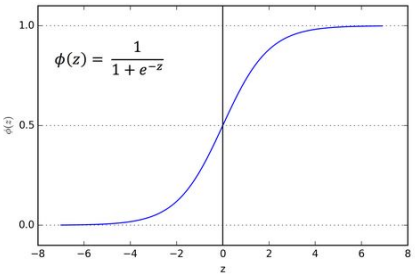
Linear and Logistic Regression



$$y = \sum w_i x_i + b$$



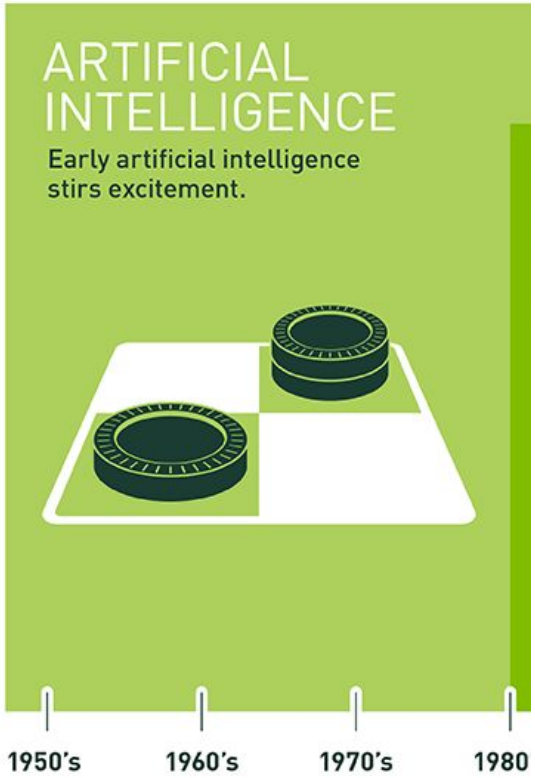
$$y = \sigma(\sum w_i x_i + b)$$



$$\sigma(z) = \frac{1}{1+e^{-z}} = sigmoid$$

FUSION

人工智慧



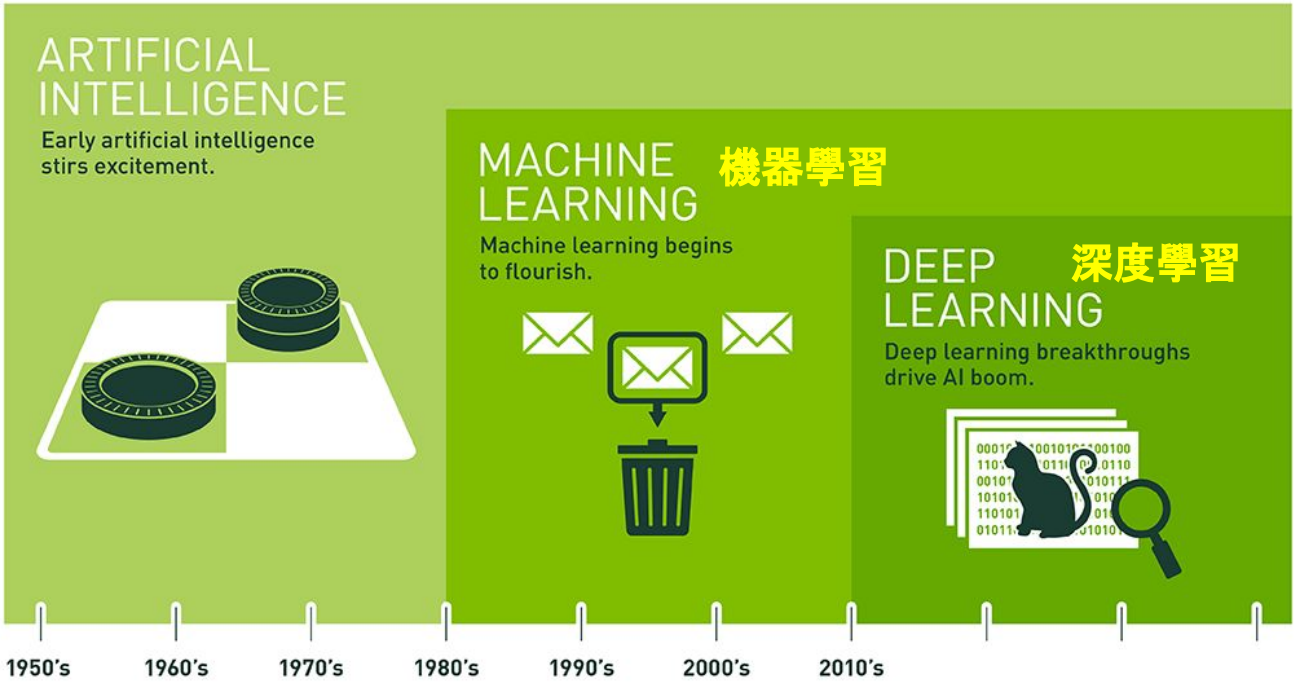
模擬和實現人類智慧的科學與技術，讓機器能夠學習、理解、推理和解決問題。

source: nVidia

人工智慧-機器學習-深度學習

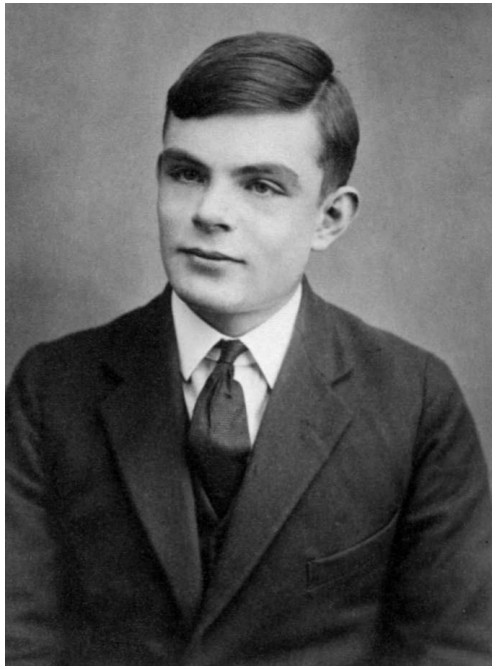
機器學習：讓機器從**資料與經驗**學習

深度學習：用**神經網路模型**做機器學習

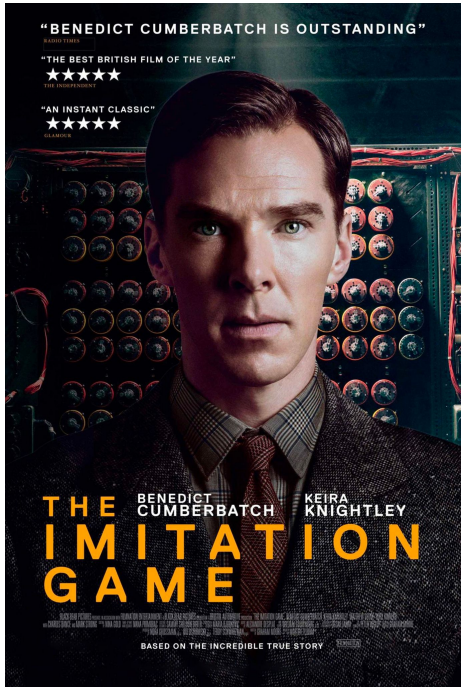


source: nVidia

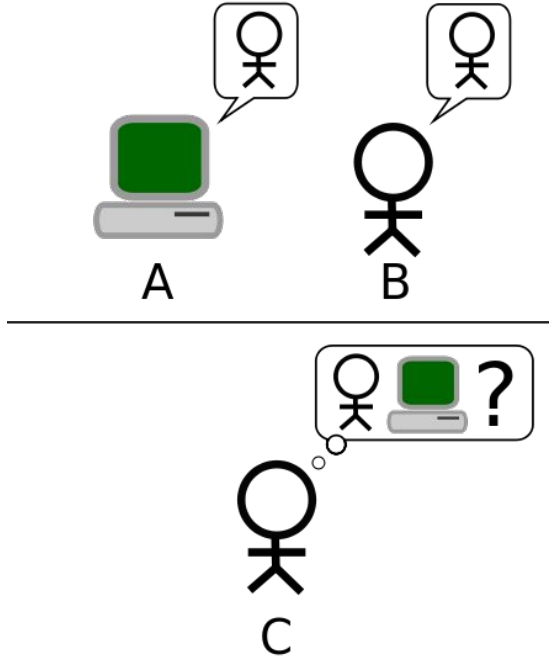
圖靈測試



Alan Turing



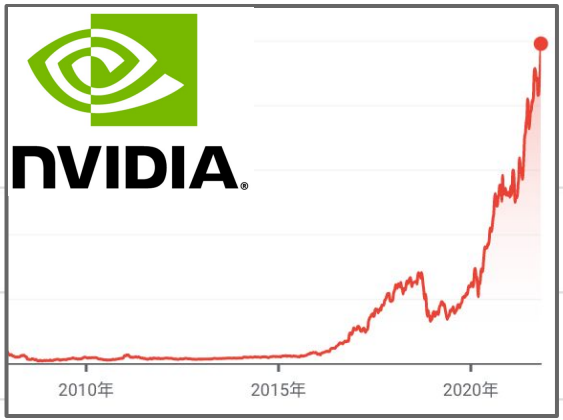
電影：模仿遊戲



如果C分不出A, B誰是機器，這台機器就可以被視為通過了圖靈測試

https://zh.wikipedia.org/wiki/%E5%9B%BE%E7%81%B5%E6%B5%8B%E8%AF%95
https://zh.wikipedia.org/wiki/%E8%89%BE%E4%BC%A6%C2%B7%E5%9B%BE%E7%81%B5

Google 趨勢：Deep Learning



AlphaGO 世紀對決

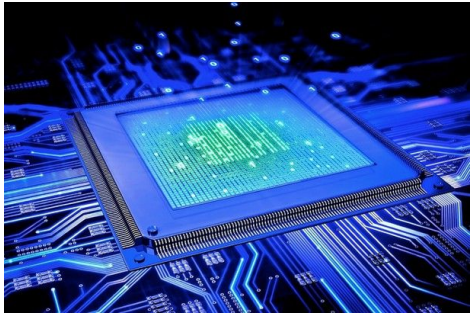
附註

2009年5月1日　2014年9月1日　2020年1月1日

FUSION

Why Now?

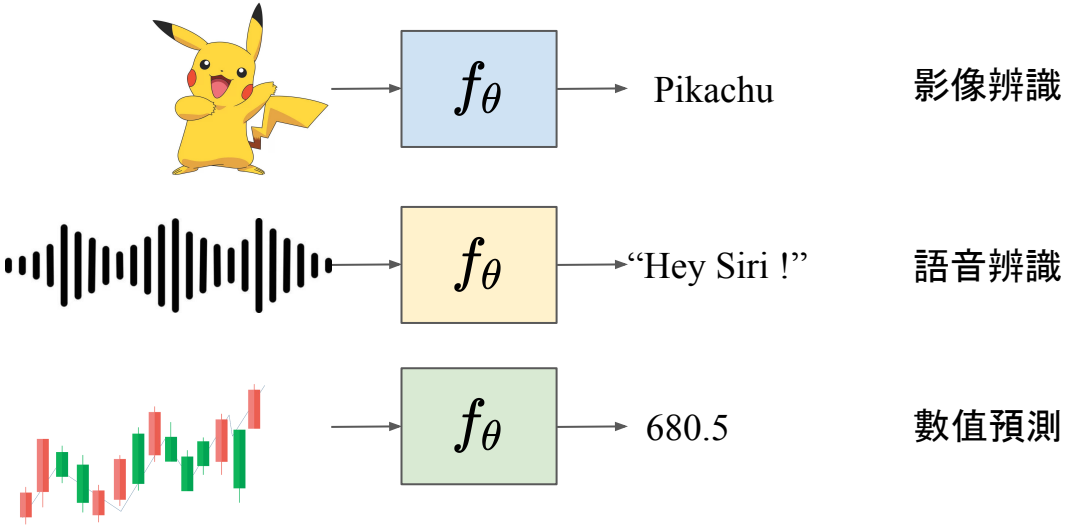
1. Hardware
   - GPU, TPU, NPU
2. **Digital** Data
3. Algorithms



FUSION

機器學習

- 從經驗中自動學習，不需要明確地編寫程式
  - To automatically learn from experience without being explicitly programmed
- 從資料中學習
  - **Learning from data**
- 找尋一個好的函式(含有參數)

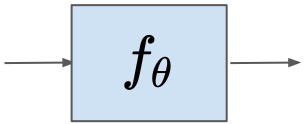


FUSION

機器學習 : 不同的輸出類型

- **回歸 (Regression)**: 預測純量 (scalar)

e.g: (骨齡預測)

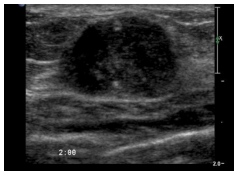


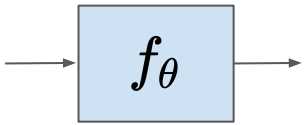
23.1

天氣溫度, 股價… etc.

- **分類 (Classification)**: 從選項中預測類別 (class)

e.g: (腫瘤診斷)





良性 / 惡性

語者辨識, 瑕疵檢測, 醫學影像分類, 文章情緒辨識… etc.

Different types of machine learning

- **Regression**: predict a **scalar**

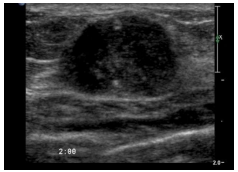
e.g: (Bone age prediction)

23.1

Weather, Stock price … etc.

- **Classification**: predict a class from **options**

e.g: (Tumor diagnosis)



Yes/No (Malignant/Benign)

Speaker, Defect, Medical Image, Article … etc.

範例 : 機器學習不同的輸出類型

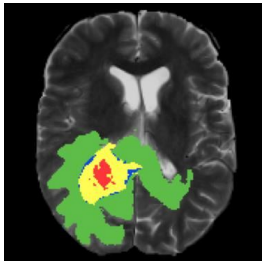
- **Examples**:

(Playing GO)

f_θ

Classification
19x19 = 361 classes

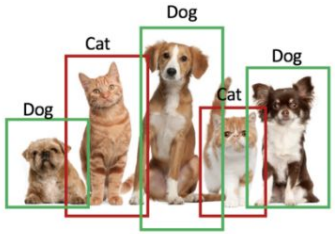
(腫瘤切割)



f_θ

Classification
每個像素做分類

(物件偵測)

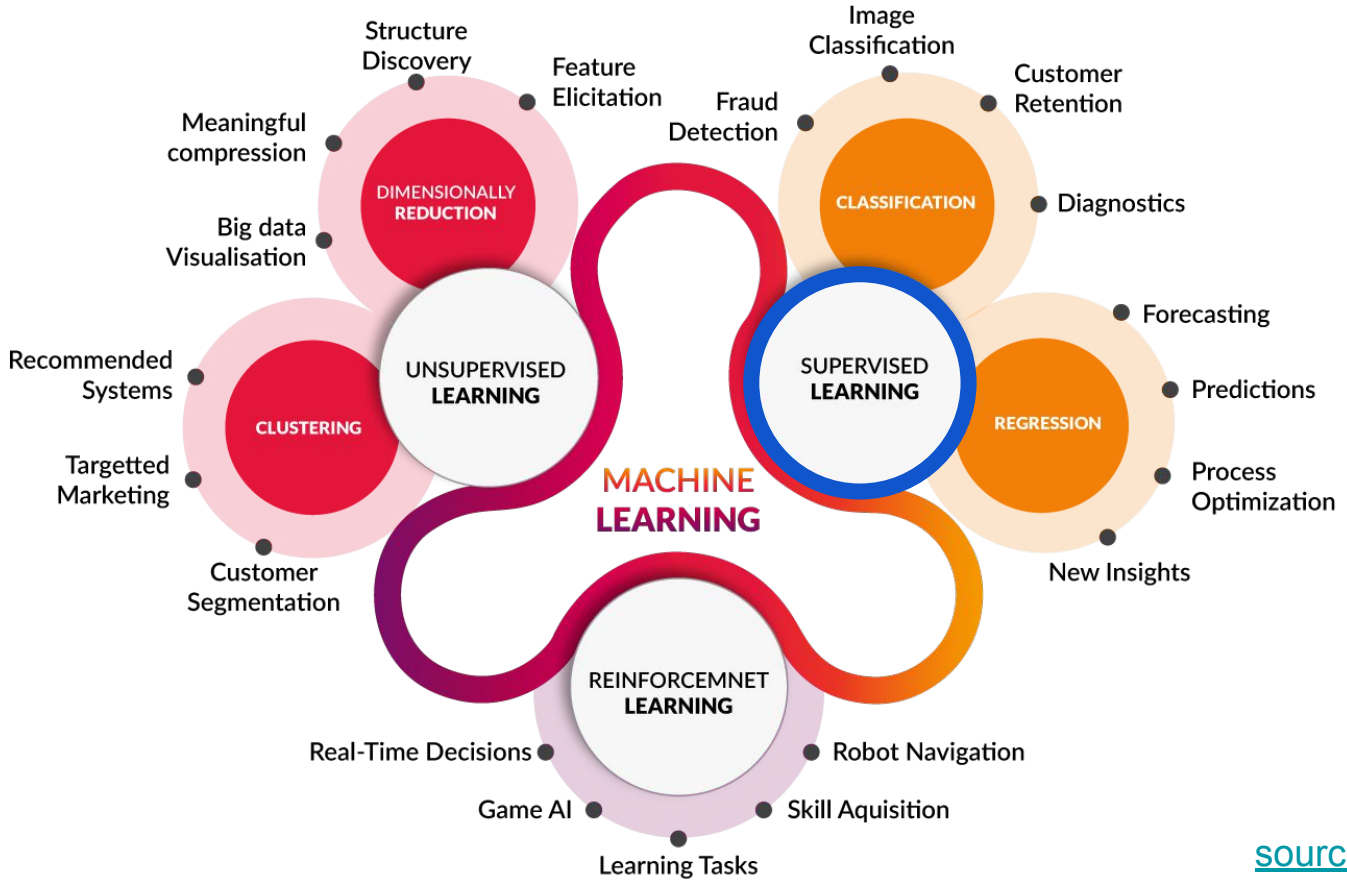


f

分類: 物件分類
回歸: 座標(x, y)、大小(w, h)

FUSION

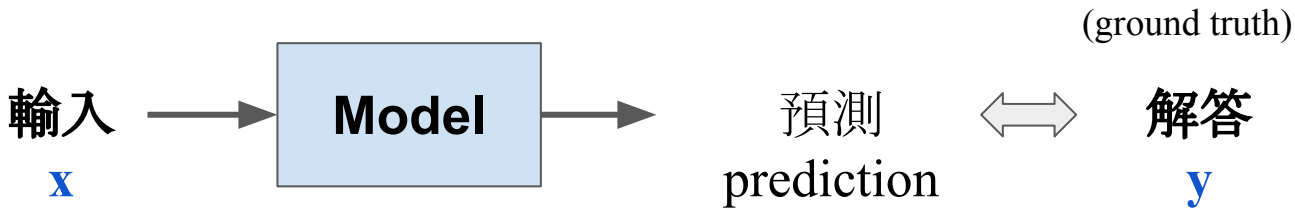
Types of Machine Learning



source

機器學習 : 學習方式

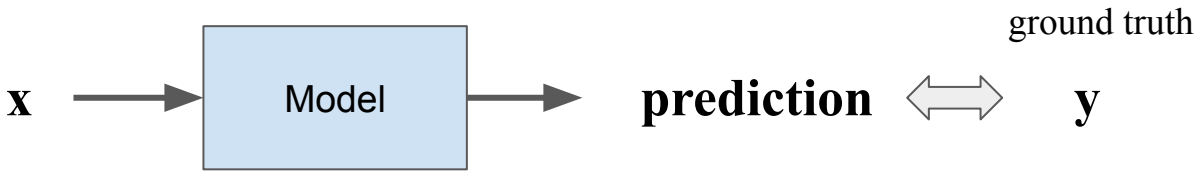
- **Supervised** Learning 監督(督導)式學習
  - 成對(X, Y): Y來自人工標註
- **Unsupervised** Learning (非監督式學習)
  - (X): Y不需要人工標註
- **Semi-supervised** Learning (半監督式學習)
  - 部分成對(X, Y)資料做監督式學習
  - 部分(X)資料做非監督式學習



source

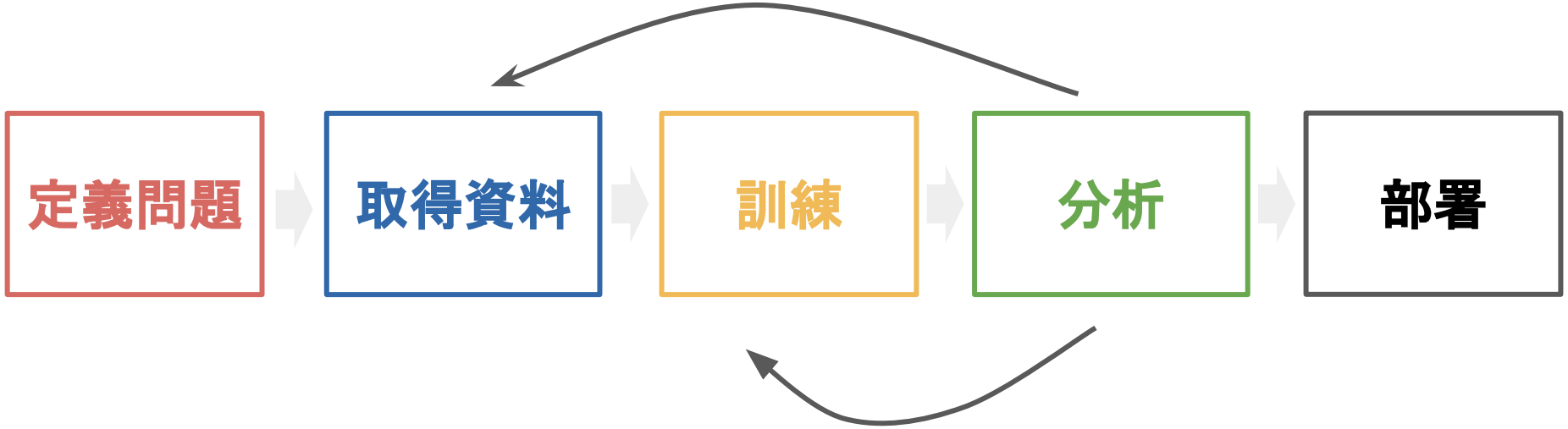
Types of Machine Learning

- **Supervised** Learning (監督式學習)
  - Paired data with ground truth: (x, y)
- **Unsupervised** Learning (非監督式學習)
  - Data **without** ground truth: (x)
- **Semi-supervised** Learning (半監督式學習)
  - Some paired data with ground truth: (x, y)
  - Some data **without** ground truth: (x)

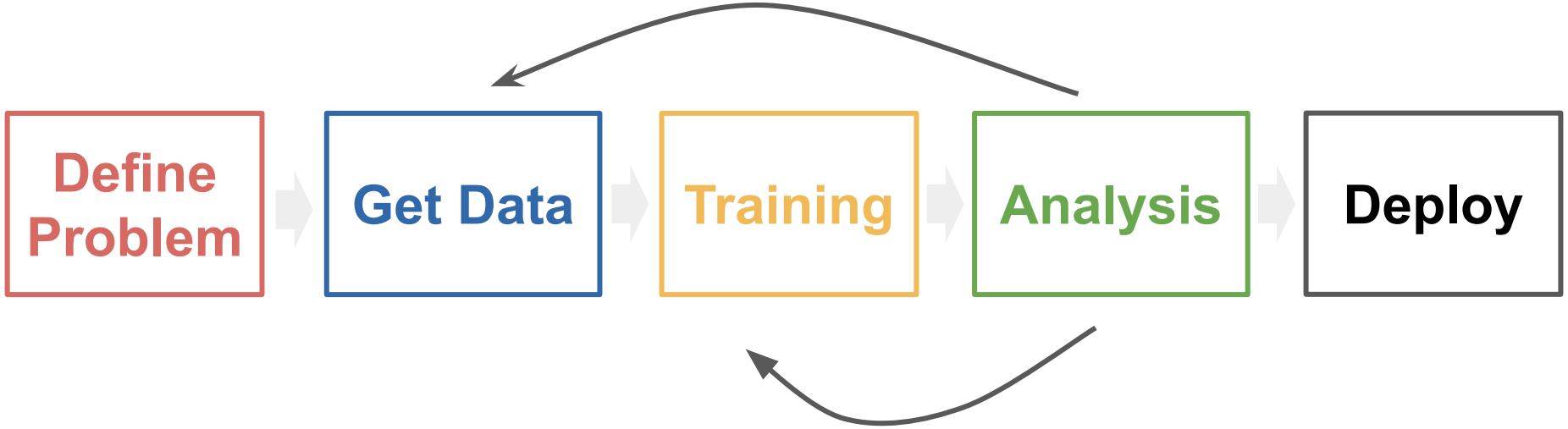


source

機器學習流程

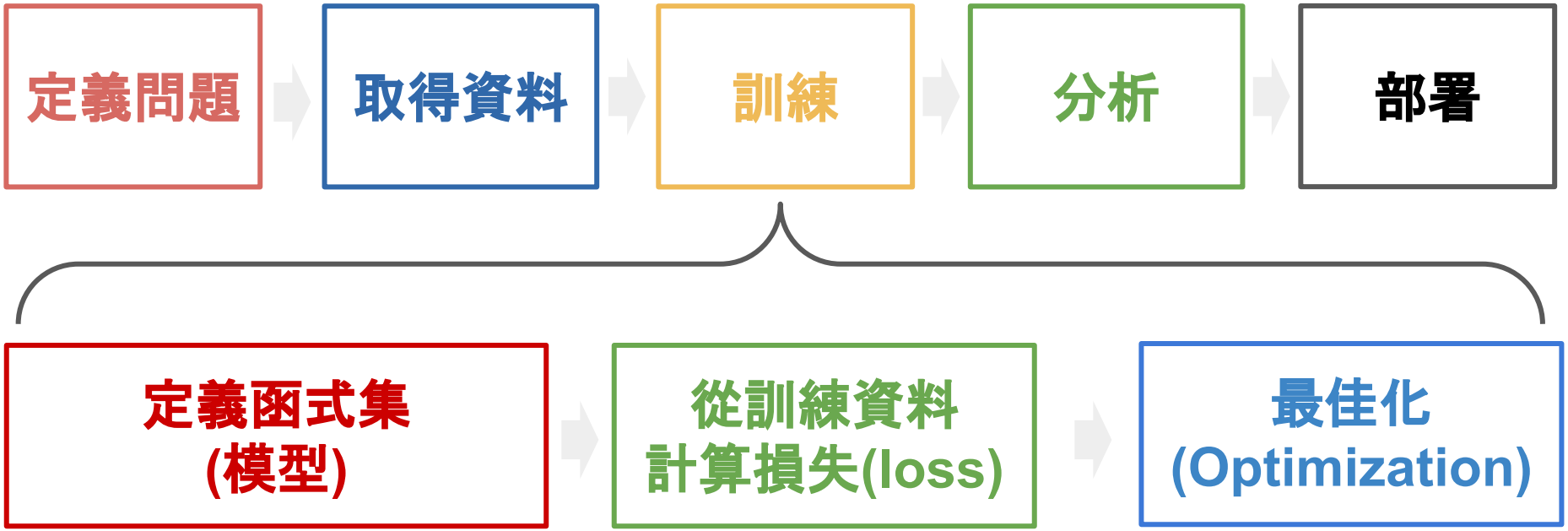


Machine learning process



FUSION

Training process



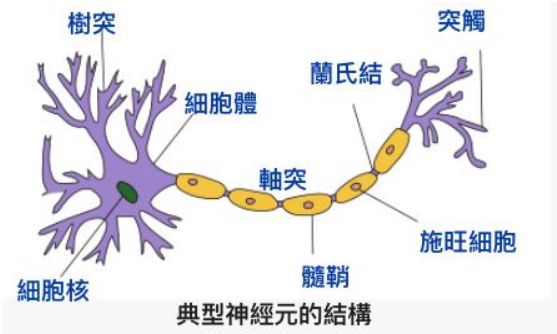
Training process

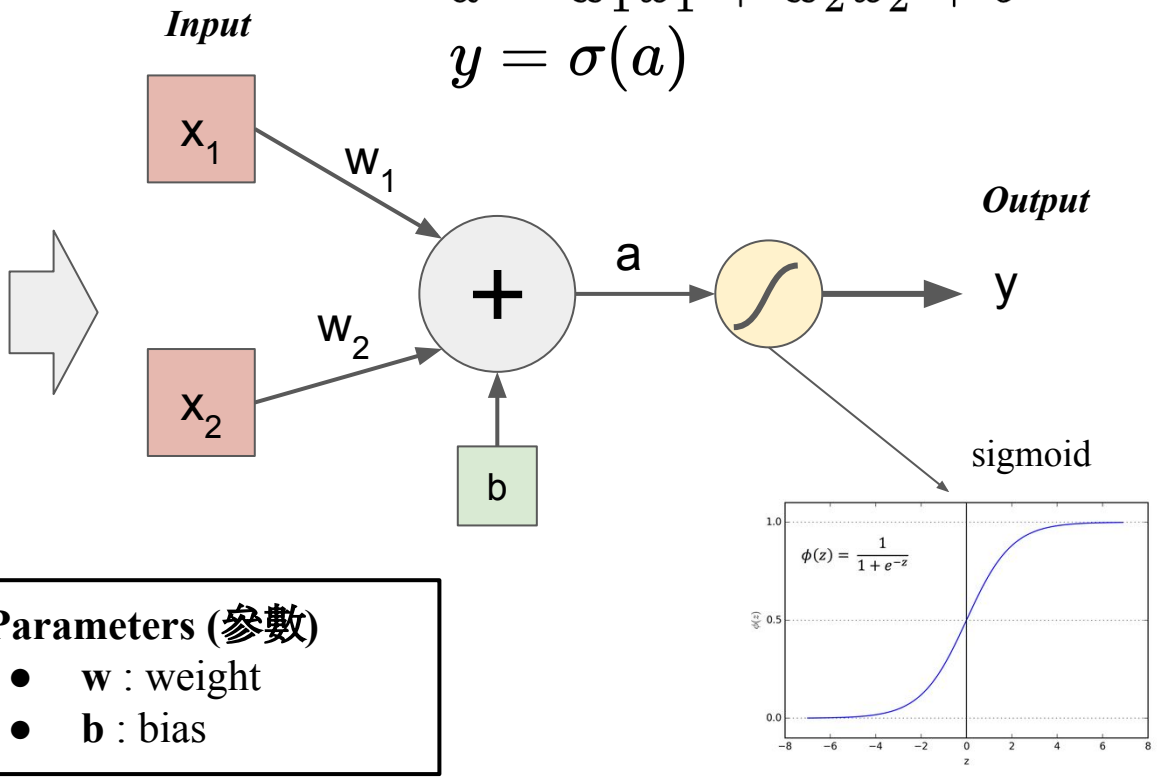
FUSION

1. Define Model: Perceptron 感知器

- 線性
- Activation function

$$a = w_1 x_1 + w_2 x_2 + b$$
$$y = \sigma(a)$$





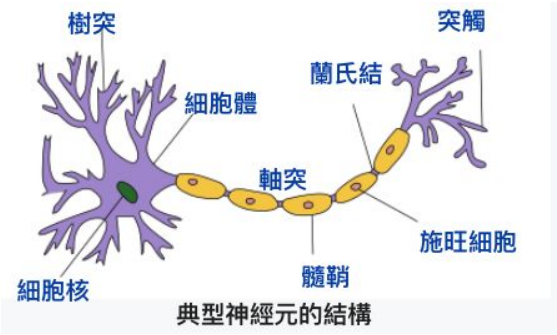
Parameters (參數)
- **w** : weight
- **b** : bias

1. 定義模型: 感知器(Perceptron)

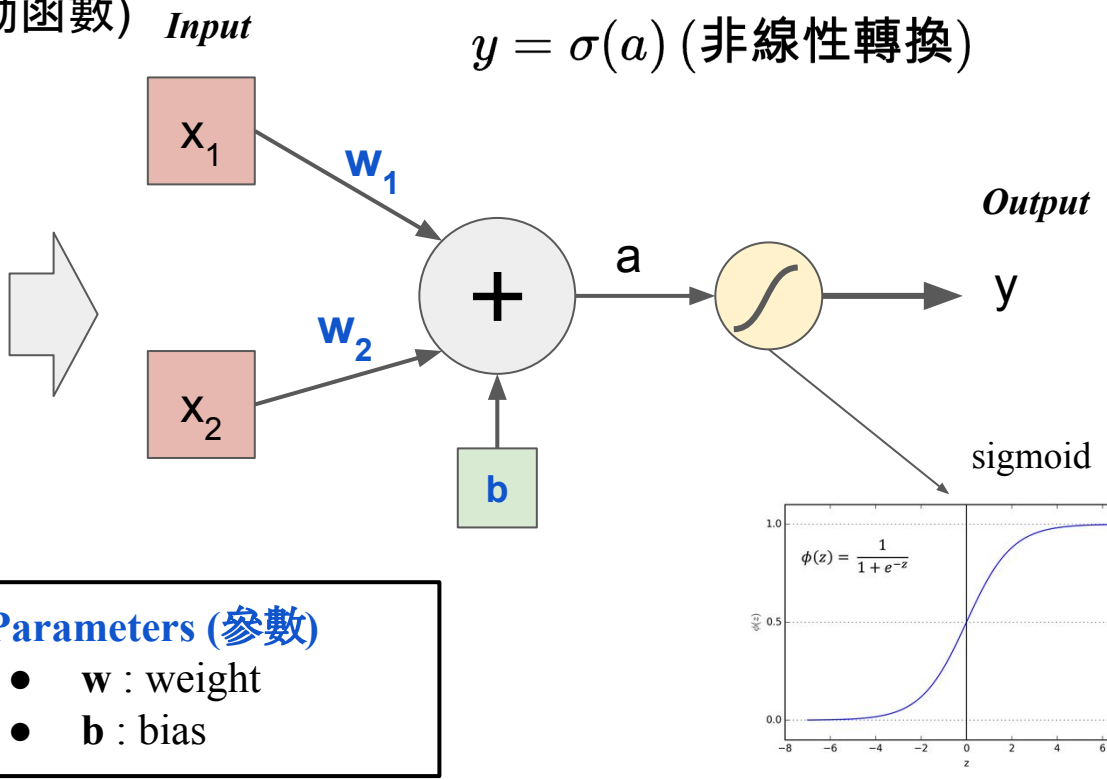
- 線性轉換
- Activation function (啟動函數)

$$a = w_1 x_1 + w_2 x_2 + b \text{ (線性轉換)}$$

$$y = \sigma(a) \text{ (非線性轉換)}$$



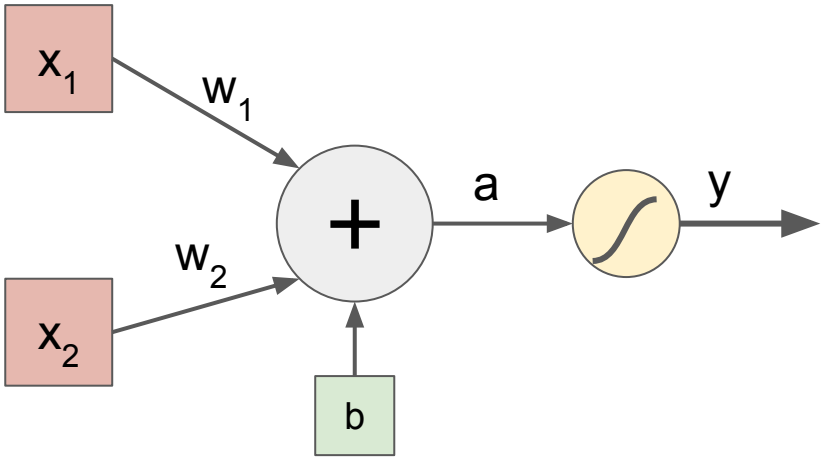
典型神經元的結構



Parameters (參數)

- **w** : weight
- **b** : bias

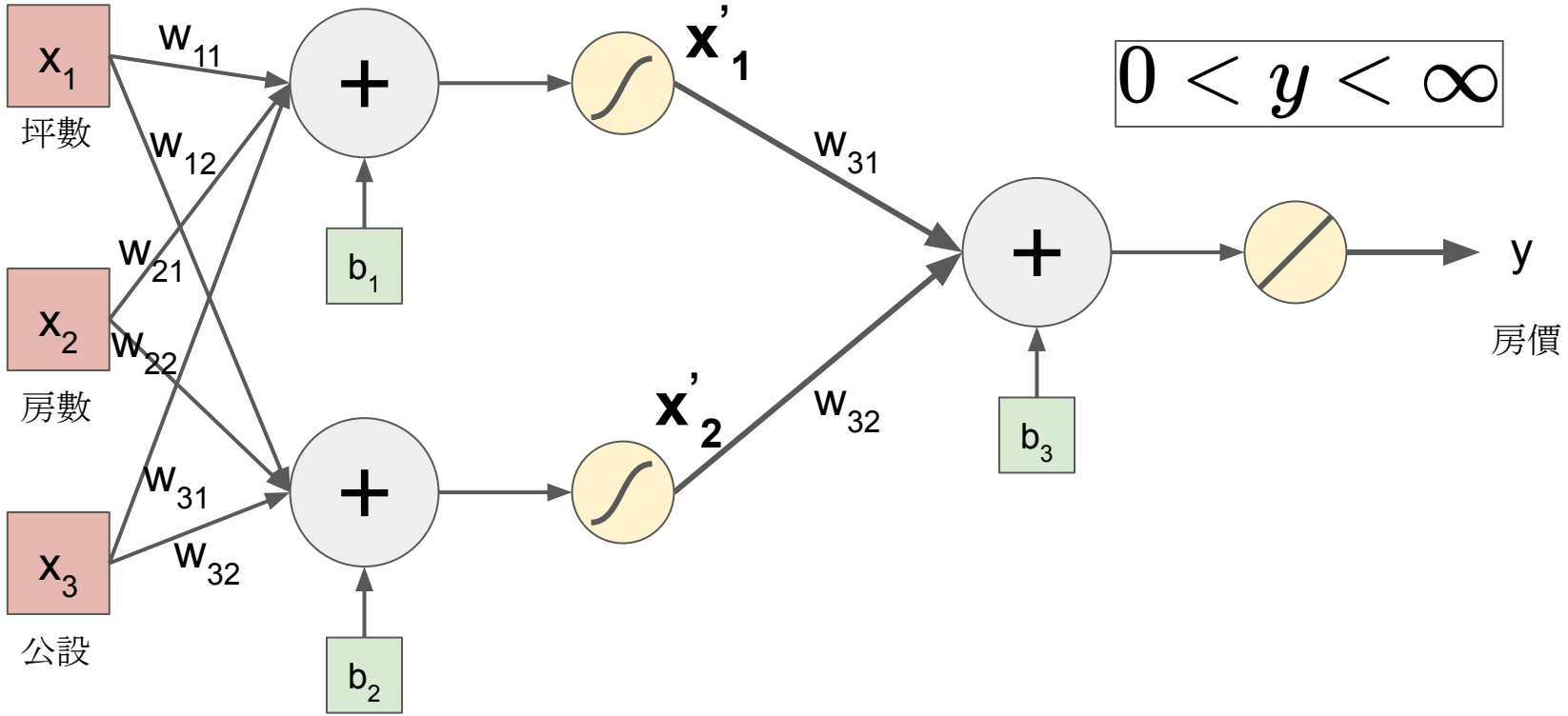
感知器 → Neural Network(神經網路)



一個不夠你有用兩個嗎？

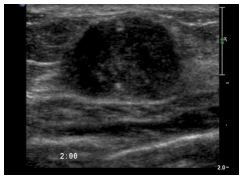
Neural Network: 神經網路 (回歸預測)

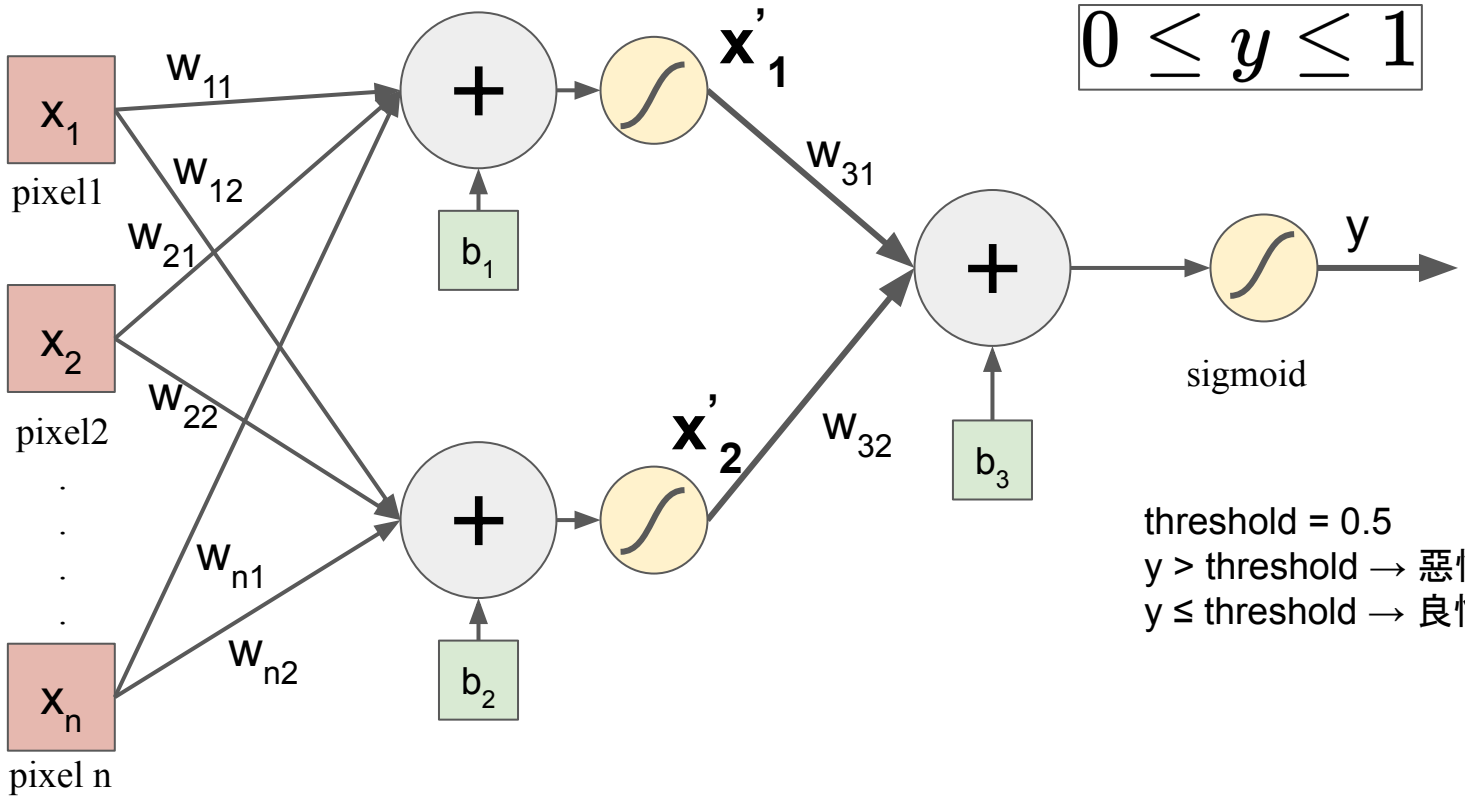
ex: 房價預測



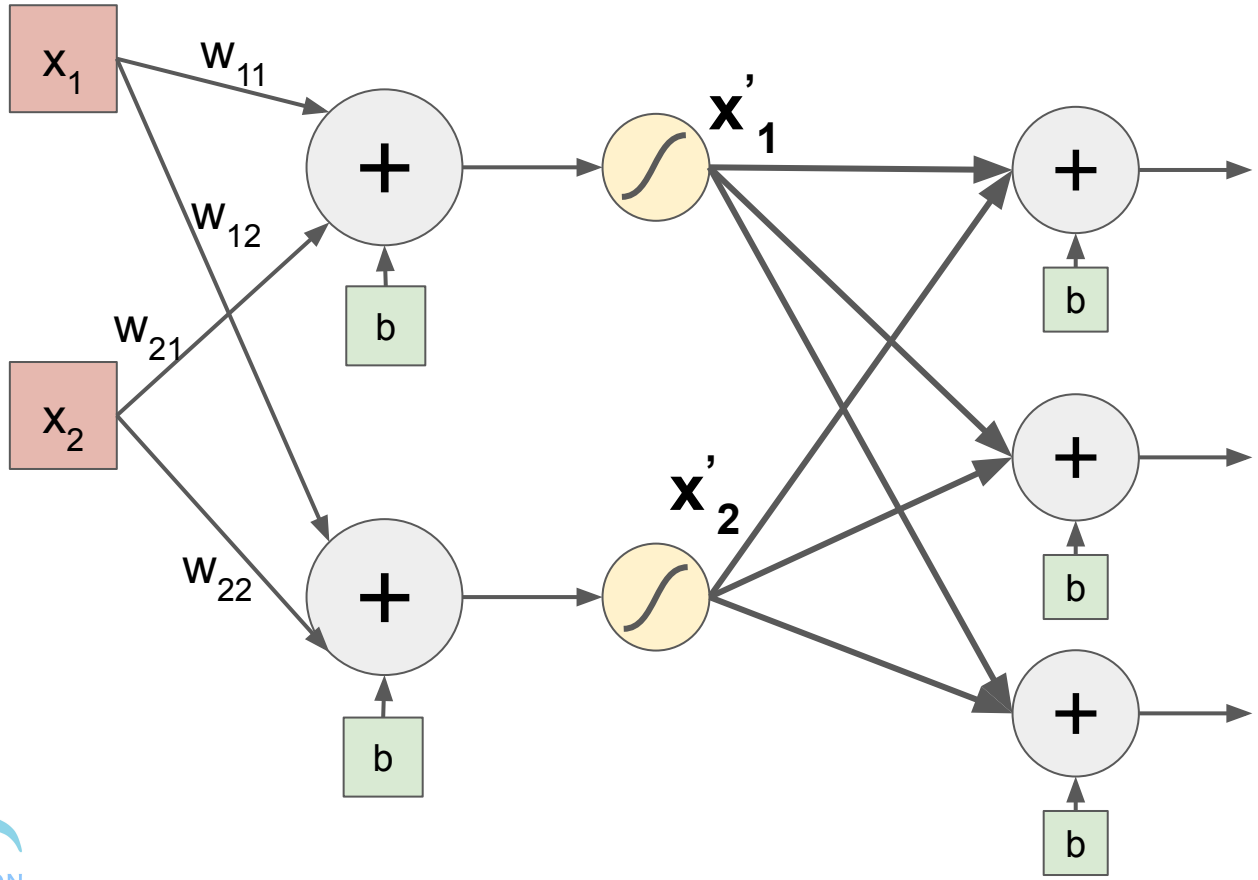
NN: Binary Classification(二元分類)

ex: 腫瘤診斷





Multi-class Classification 多元分類



logits

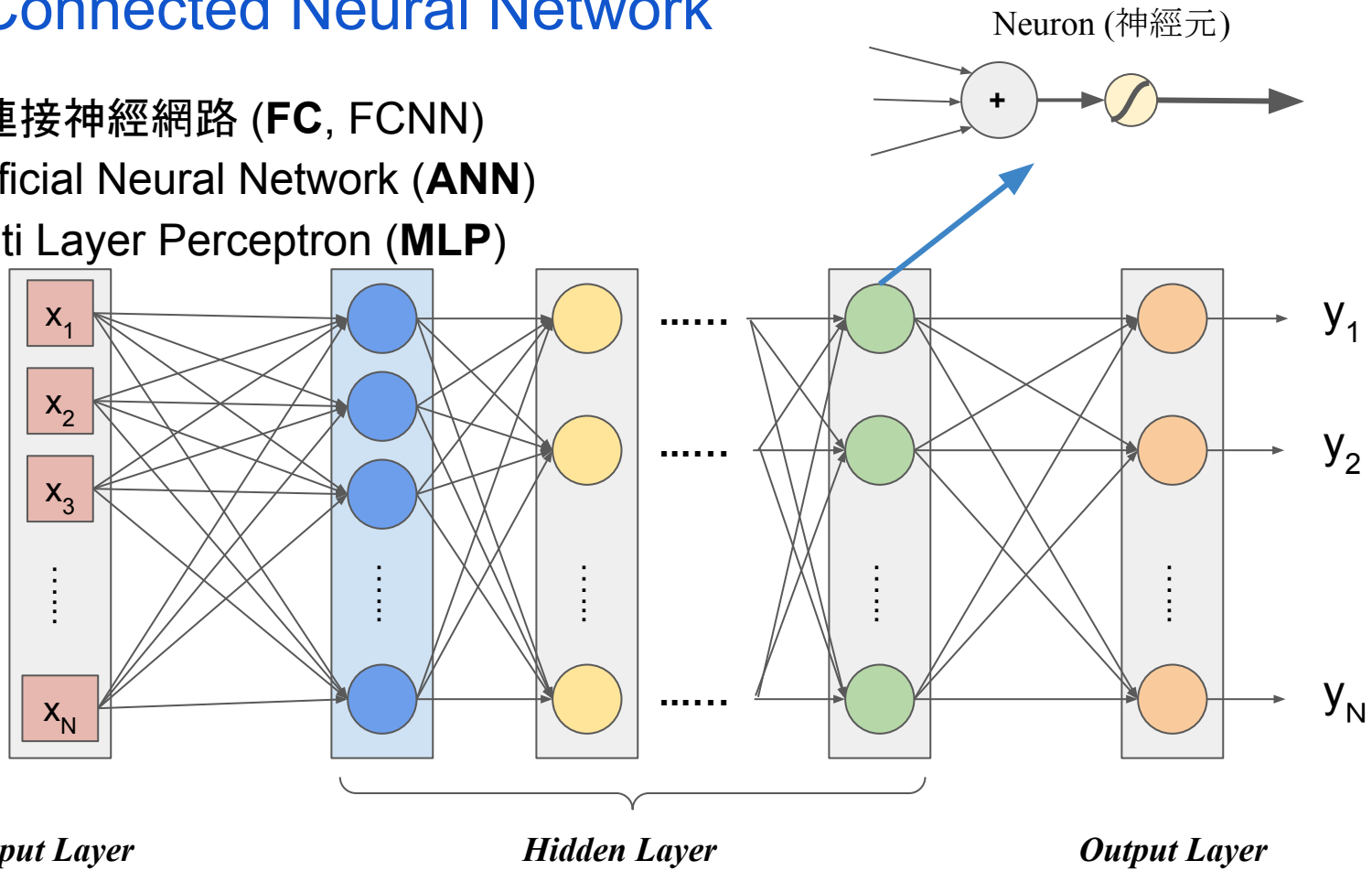
$$-\infty < y_i < \infty$$

預測類別 = argmax(y_i)

(0 or 1 or 2)

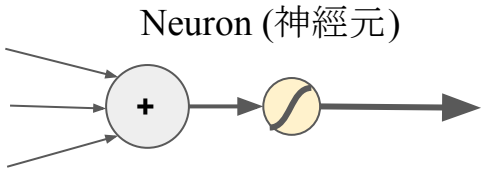
Fully Connected Neural Network

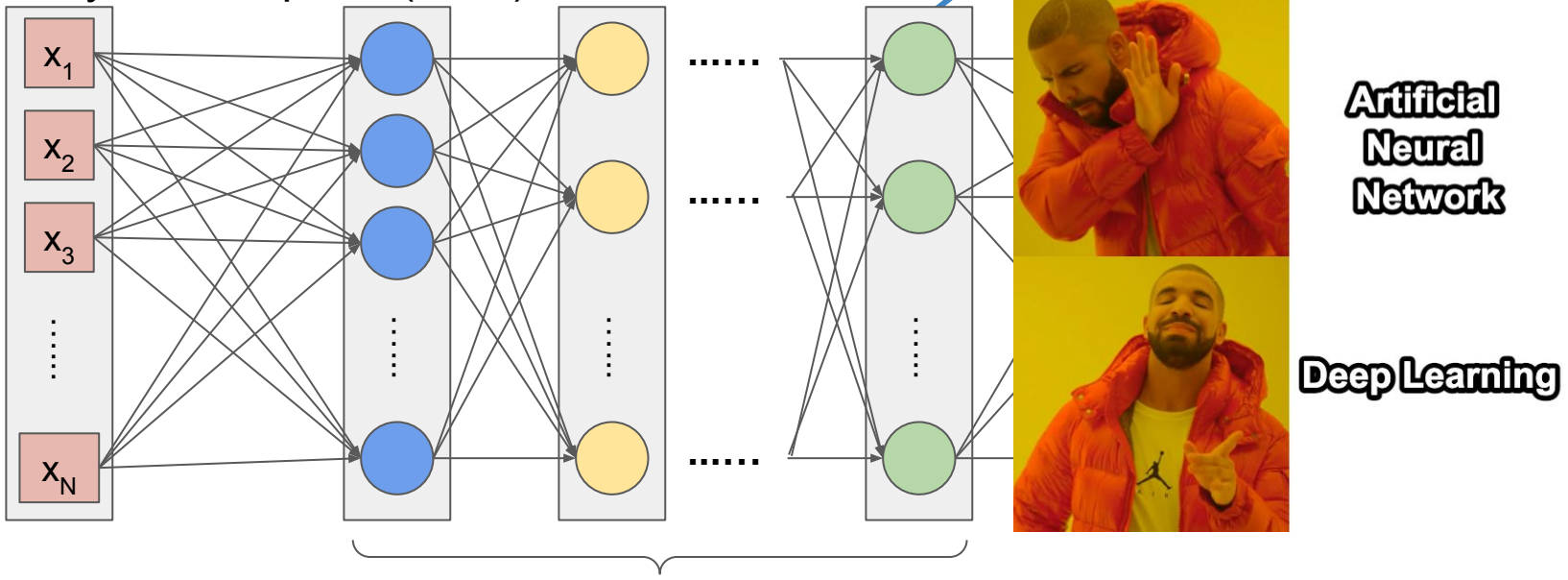
- 全連接神經網路 (**FC**, FCNN)
- Artificial Neural Network (**ANN**)
- Multi Layer Perceptron (**MLP**)



Fully Connected Neural Network

- 全連接神經網路 (**FC**, FCNN)
- Artificial Neural Network (**ANN**)
- Multi Layer Perceptron (**MLP**)

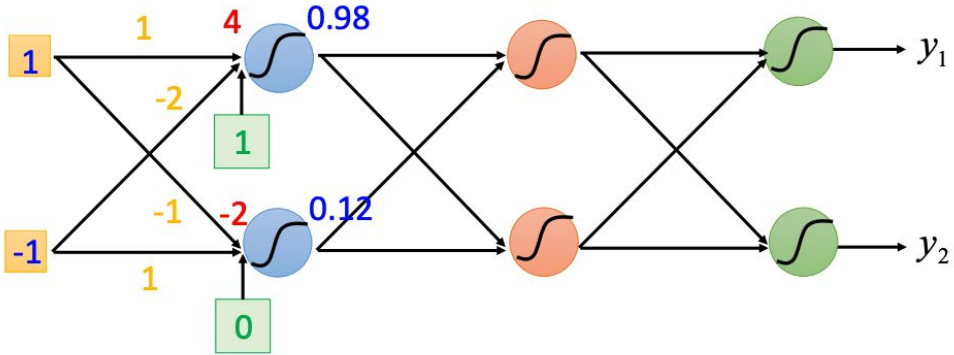




FUSION

為什麼需要 GPU？

- 平行化加速矩陣運算

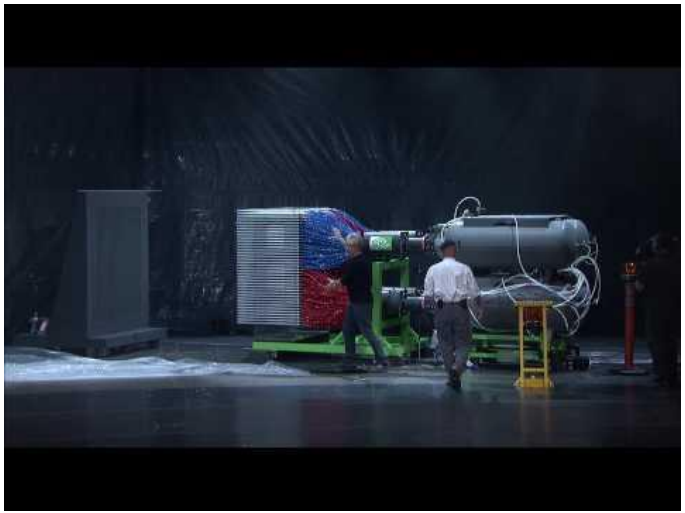


weight input bias output

$$\sigma\left(\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}\right)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

(2, **2**) $\begin{bmatrix}4\\ -2\end{bmatrix}$ (**2**)

GPU v.s. CPU

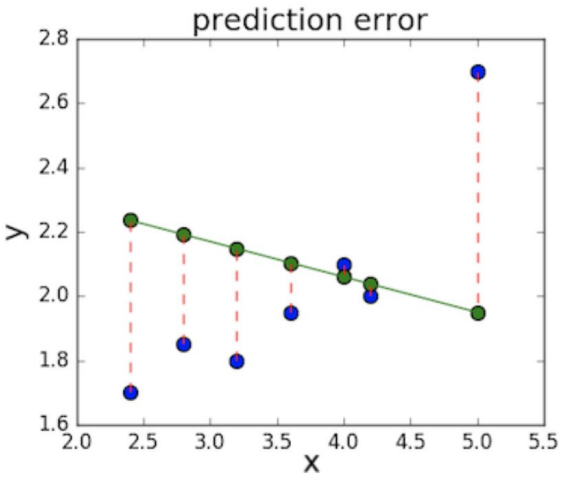


https://www.youtube.com/watch?v=-P28LKWTzrI

source: 李宏毅DL

2. Get loss from training data

- Loss function (損失函數)
  - What is Good?
  - The **error** between **prediction** and **truth**



Regression loss function

MSE
(Mean square error)

$$\frac{\sum (y-\hat{y})^2}{N}$$

MAE
(Mean absulute error)

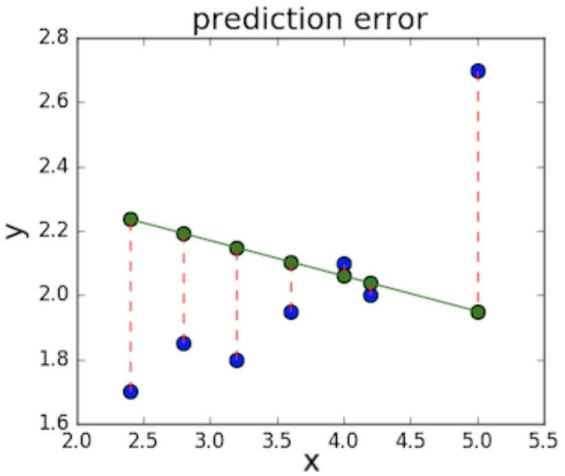
$$\frac{\sum |y-\hat{y}|}{N}$$

y : prediction
$\hat{y}$: answer
N : number of samples

FUSION

2. Get loss from training data

- Loss function (損失函數)
  - What is Good?
  - 計算**預測值**與**解答**的**誤差**



Regression loss function

MSE
(Mean square error)

$$\frac{\sum (y-\hat{y})^2}{N}$$

MAE
(Mean absulute error)

$$\frac{\sum |y-\hat{y}|}{N}$$

y : prediction
$\hat{y}$: answer
N : number of samples

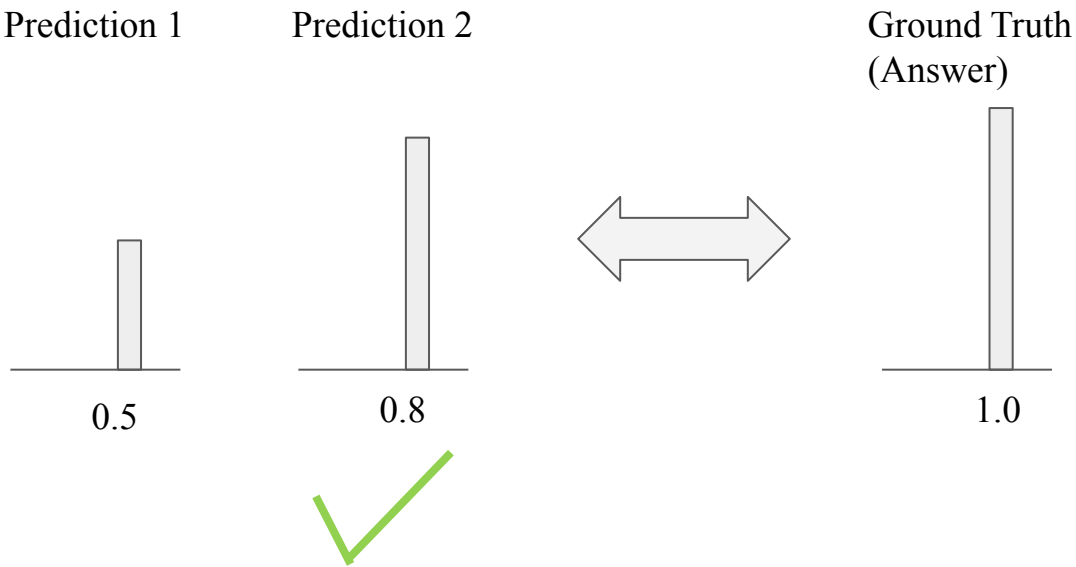
FUSION

Loss Function: Binary Classification

- Binary Cross Entropy (BCE)

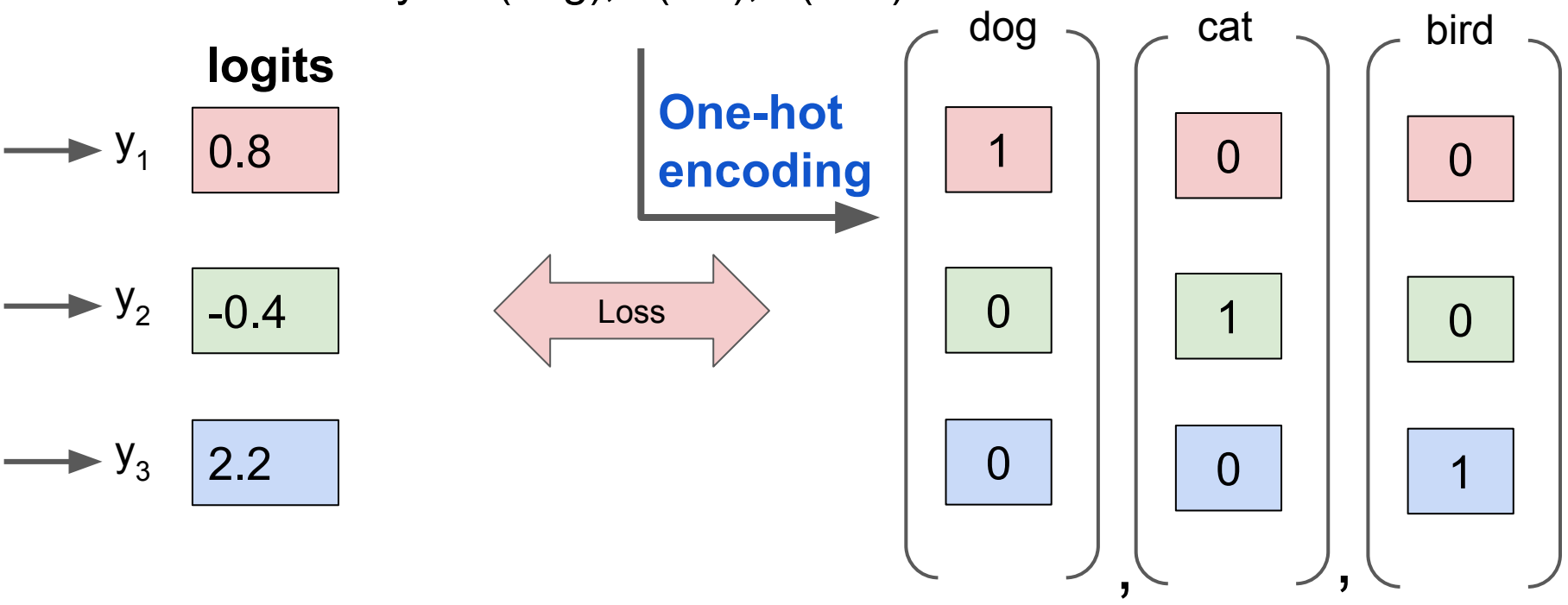
$$BCE = -\frac{1}{N}\sum_{i=1}^{N} \hat{y}_i log y_i + (1-\hat{y}_i) log(1-y_i)$$

$$y : 0 \sim 1$$
$$\hat{y} : 0, 1$$



Multi-class Classification 多元分類

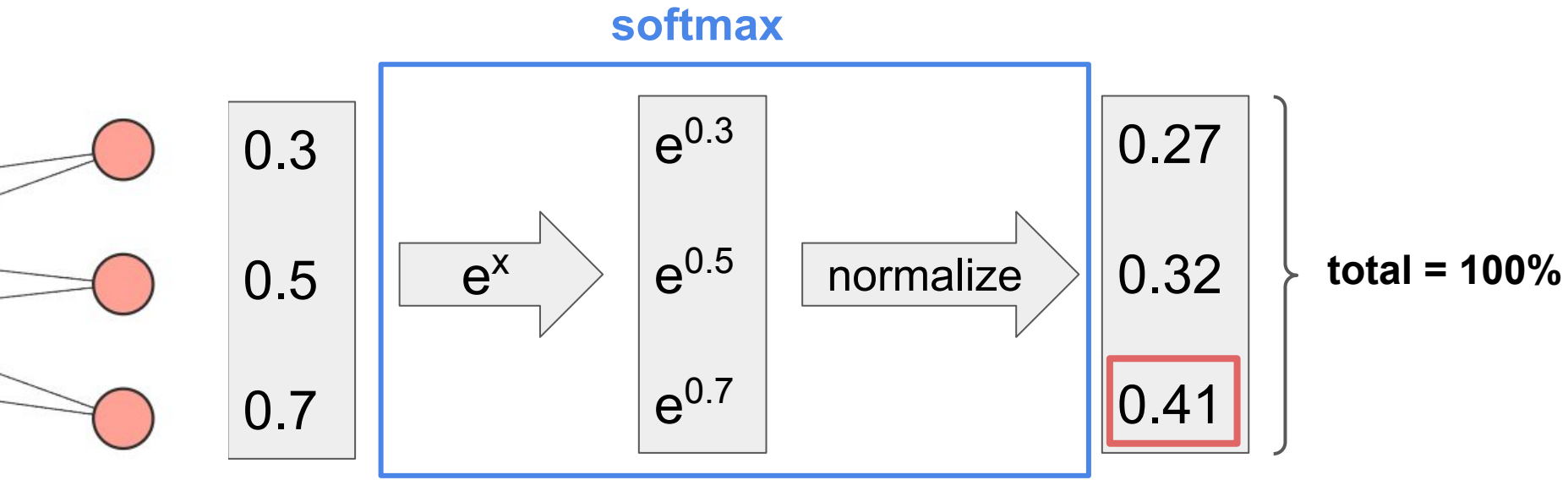
y = 0(dog), 1(cat), 2(bird)



Prediction

Label (Ground Truth)

Softmax (For human reading)



$$softmax(z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

FUSION

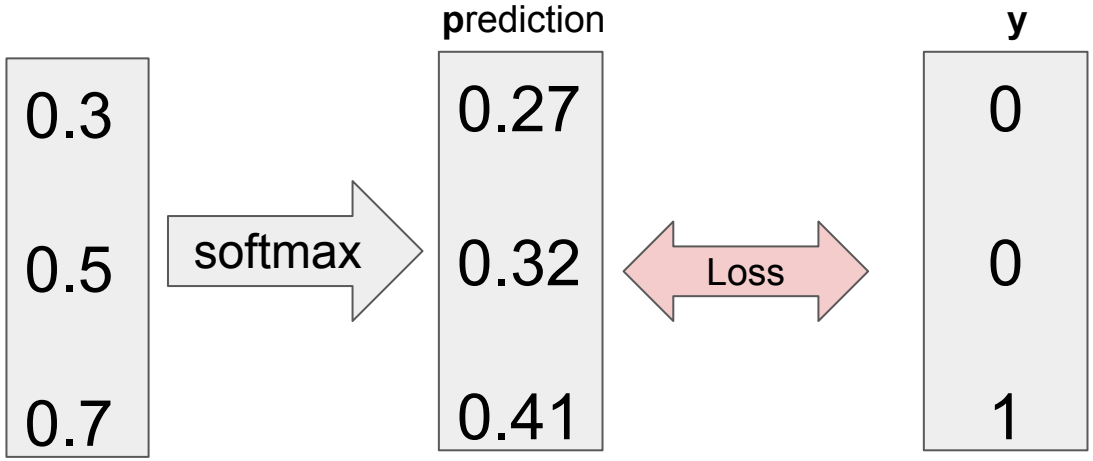
Loss function: Multi-class Classification

- Cross Entropy

$$CCE = -\frac{1}{N}\sum \hat{y} log_e y$$

N: batch size

y : prediction
$\hat{y}$: answer
N : number of samples



-**Σy_true** $\log_e$(**y_pred**)
= - (0×$\log_e$(0.27) + 0×$\log_e$(0.32)+ 1×$\log_e$(0.41))
= 0.89 ↓

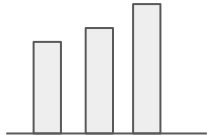
Cross Entropy

prediction

0.27

0.32

0.41



Loss

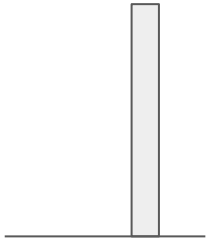
ground truth

y

0

0

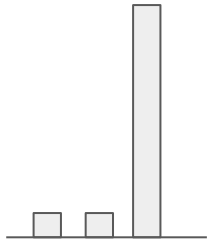
1



0.05

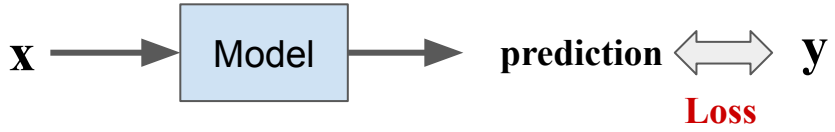
0.05

0.9



FUSION

3. Optimization



- model with **1 parameter**

Gradient Decesnt : 梯度下降

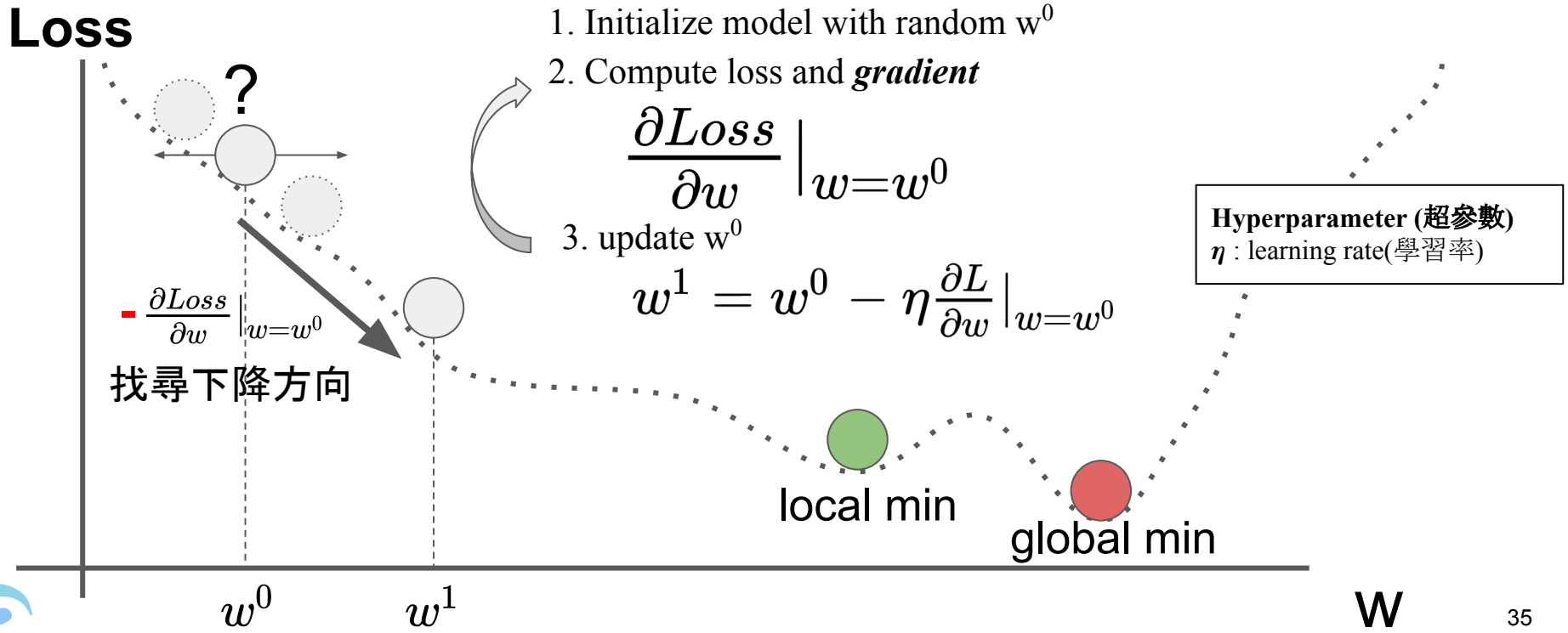
1. Initialize model with random w^0
2. Compute loss and ***gradient***

$$\frac{\partial Loss}{\partial w}\Big|_{w=w^0}$$

3. update w^0

$$w^1 = w^0 - \eta \frac{\partial L}{\partial w}\Big|_{w=w^0}$$

Hyperparameter (超參數)
η : learning rate(學習率)



3. Optimization

Goal : $w^*, b^* = arg \min_{w,b} L$

Goal: 找到Loss最小的參數(w, b)

Gradient Decesnt : 梯度下降

1. Initializa with random value w^0 , b^0
2. Compute ***gradient***
3. Update w, b

$$w^1 = w^0 - \eta \frac{\partial L}{\partial w}\Big|_{w=w^0, b=b^0}$$

$$b^1 = b^0 - \eta \frac{\partial L}{\partial b}\Big|_{w=w^0, b=b^0}$$

FUSION

3. Optimization

Goal : $\theta^* = arg\min_{\theta} L$

$$\theta = \begin{bmatrix} w_0 \\ w_1 \\ b_0 \\ \vdots \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \end{bmatrix}$$ (模型所有參數)

Gradient Decesnt : 梯度下降

1. Initializa with random ***θ***
2. Compute ***gradient***
3. Update ***θ***

Goal: 找到Loss最小的參數θ

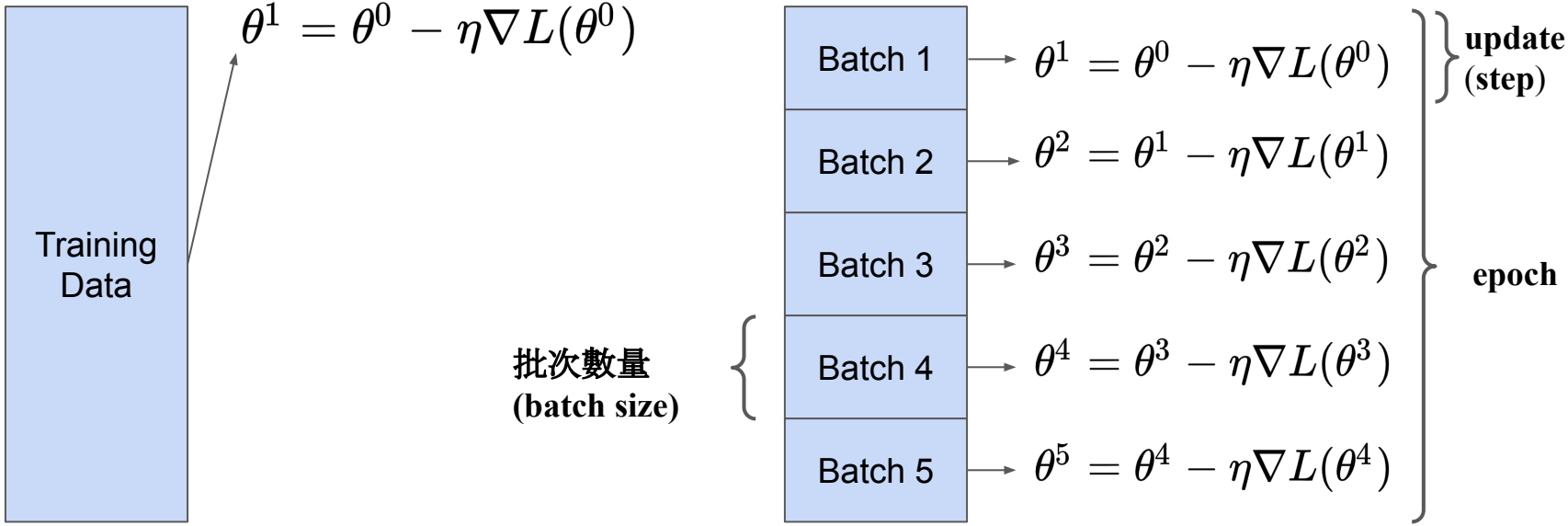
$$G = \begin{bmatrix} \frac{\partial L}{\partial \theta_0}|_{\theta=\theta^0} \\ \frac{\partial L}{\partial \theta_1}|_{\theta=\theta^0} \\ \vdots \end{bmatrix} \qquad \begin{bmatrix} \theta_0^1 \\ \theta_1^1 \\ \vdots \end{bmatrix} = \begin{bmatrix} \theta_0^0 \\ \theta_1^0 \\ \vdots \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial L}{\partial \theta_0}|_{\theta=\theta^0} \\ \frac{\partial L}{\partial \theta_1}|_{\theta=\theta^0} \\ \vdots \end{bmatrix}$$

$$G = \nabla L(\theta^0) \qquad \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$
$$= \theta^0 - \eta G$$

FUSION

Training - 資料批次化

1. 初始化模型參數 θ^0
2. 更新模型參數 θ

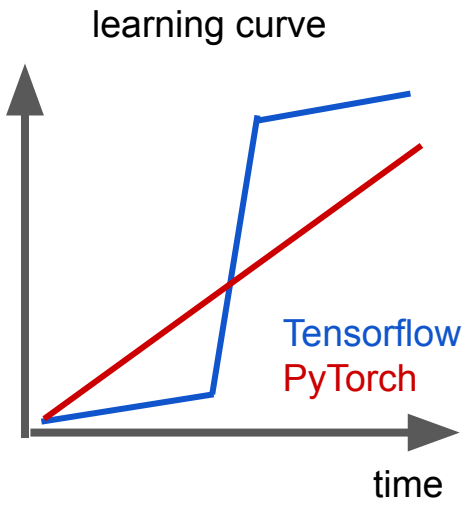


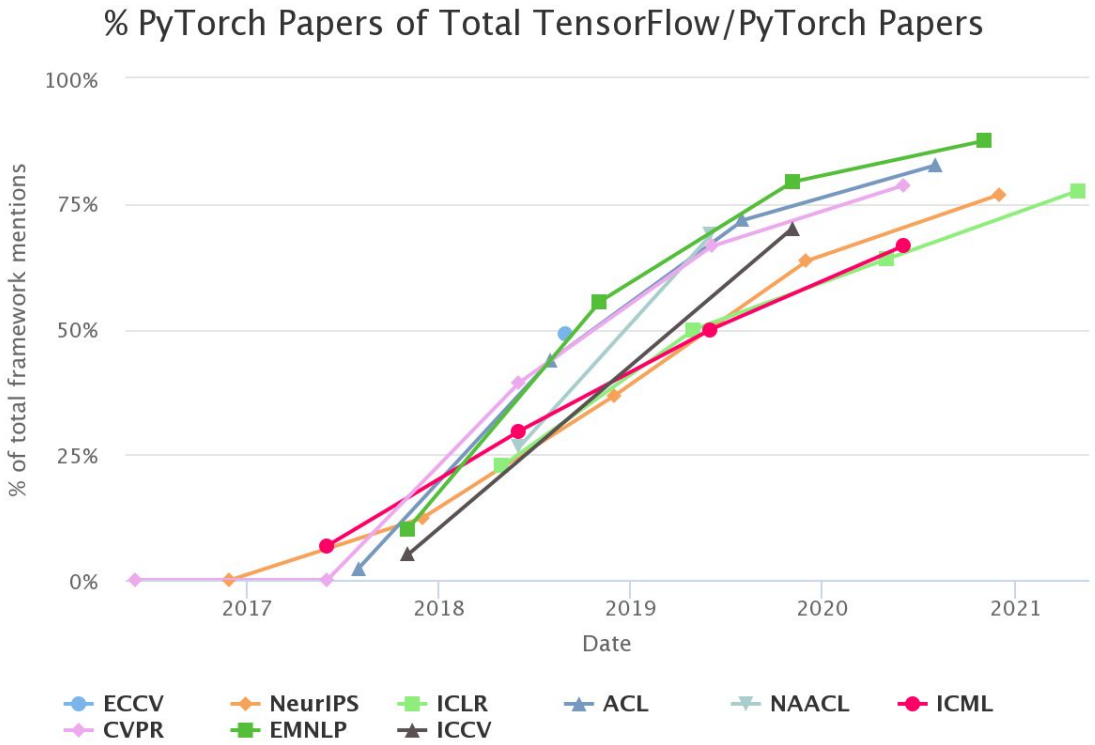
FUSION

https://pytorch.org/

PyTorch

- https://pytorch.org/
- Why PyTorch?



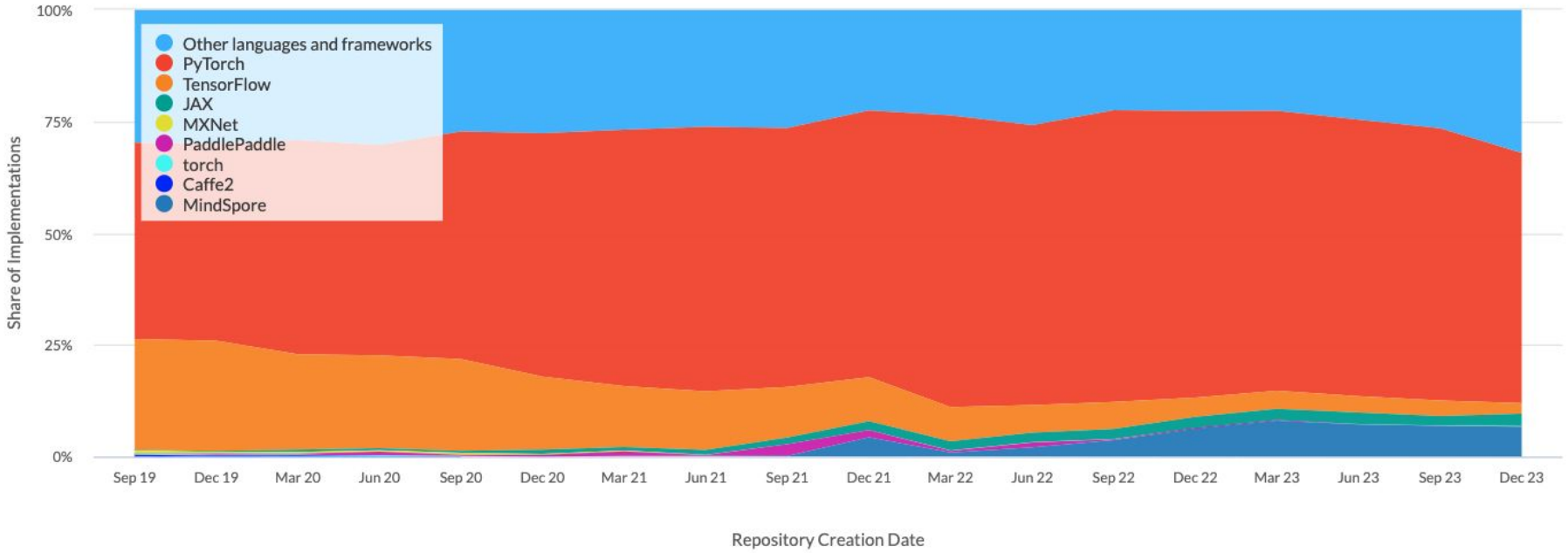


http://horace.io/pytorch-vs-tensorflow/

FUSION

Trends

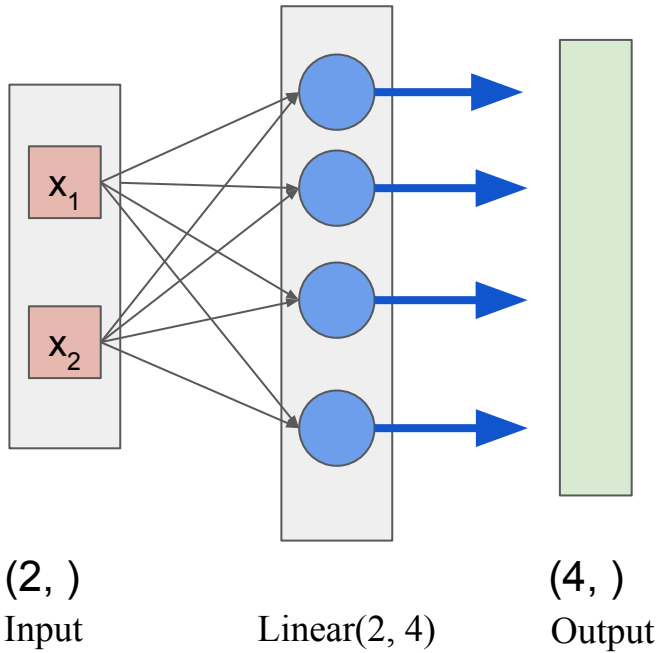
https://paperswithcode.com/trends



FUSION

torch.nn.Linear

- torch.nn.Linear(in_features, out_features)



PyTorch

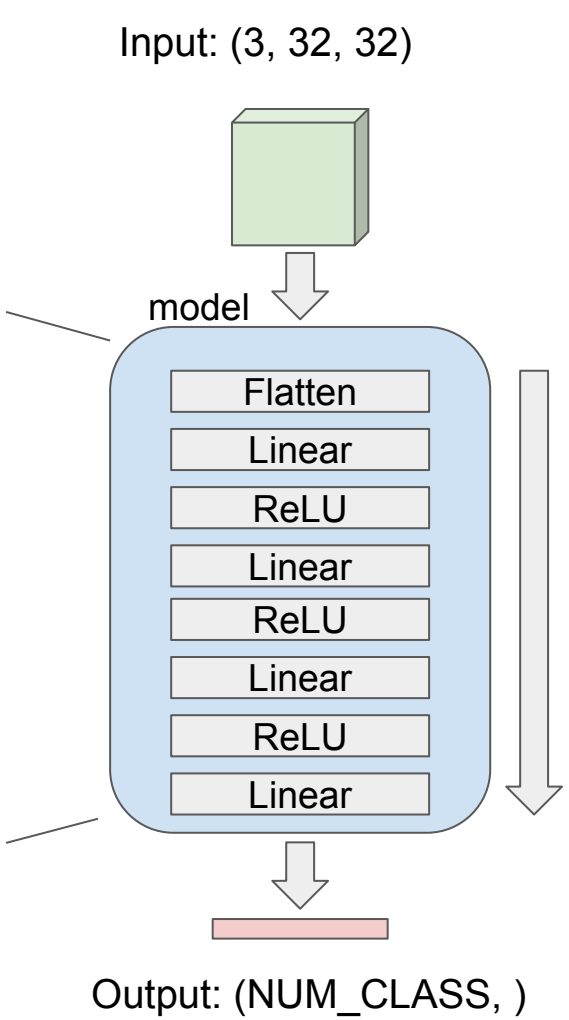
```
nn.Linear(
    in_features=2,
    out_features=4
)
```

torch.nn.Sequential

序列化模型: 網路層一層一層按順序堆疊

PyTorch

```
model = nn.Sequential(
    nn.Flatten(), # (3, 32, 32) -> (3*32*32, )
    nn.Linear(in_features=IMG_SIZE*IMG_SIZE*3,
              out_features=64), # (C*H*W) -> (64)
    nn.ReLU(),
    nn.Linear(64, 128), # (64) -> (128)
    nn.ReLU(),
    nn.Linear(128, 128),
    nn.ReLU(),
    nn.Linear(128, NUM_CLASS), # (128) -> NUM_CLASS
)
```



FUSION

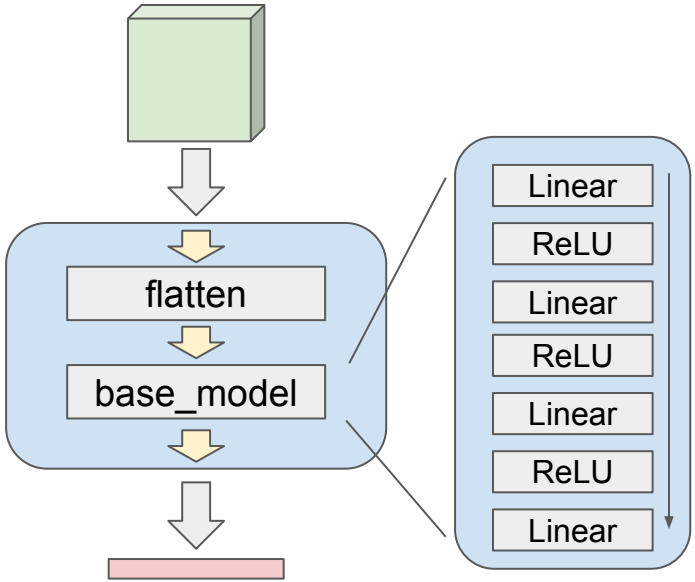
torch.nn.Module

PyTorch

```
# 繼承 nn.Module
class NeuralNet(nn.Module):
    def __init__(self):
        # 網路層初始化
        super().__init__()
        self.flatten = nn.Flatten()
        self.base_model = nn.Sequential(
            nn.Linear(IMG_SIZE*IMG_SIZE*3, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 16),
            nn.ReLU(),
            nn.Linear(16, NUM_CLASS)
        )
```

```
model = NeuralNet() # 創建模型
```

Input: (3, 32, 32)



Output: (NUM_CLASS,)

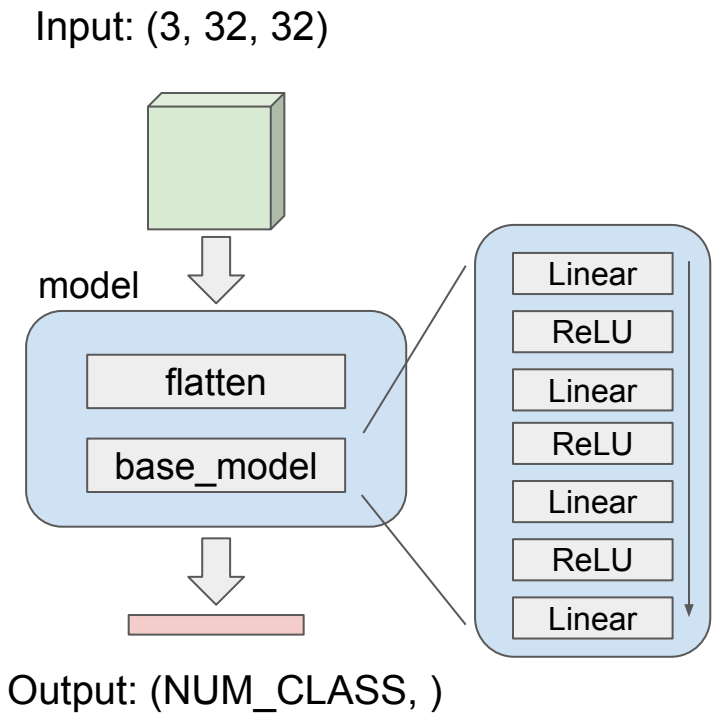
FUSION

torch.nn.Module

PyTorch

```python
class NeuralNet(nn.Module):
    def __init__(self):
        # ...
    def forward(self, x):
        # forward函式定義輸入的資料張量該用如何運算
        # 輸入資料x: (bs, 3, 32, 32)
        # 輸入模型都是 torch.Tensor
        # 且都帶有批次數量的維度 batch_size

        x = self.flatten(x)
        # flatten: (bs, 3, 32, 32) -> (bs, 3072)

        logits = self.base_model(x)
        # base_model: (bs, 3072) -> (bs, 10)

        return logits
```

```python
model = NeuralNet() # 創建模型
```



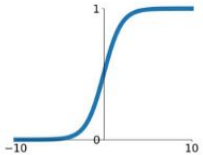
FUSION

torch.nn Non-linear Activations

- nn.ReLU()
- nn.Sigmoid()
- nn.LeakyReLU()
- ...

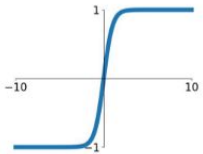
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



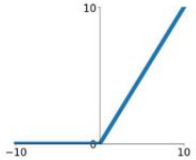
tanh

$$\tanh(x)$$



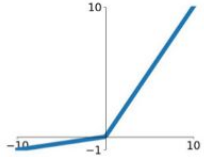
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

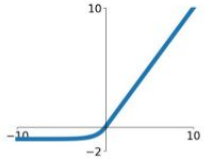


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



FUSION

Model Summary

```
print(model)
```

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=3072, out_features=64, bias=True)
  (2): ReLU()
  (3): Linear(in_features=64, out_features=128, bias=True)
  (4): ReLU()
  (5): Linear(in_features=128, out_features=128, bias=True)
  (6): ReLU()
  (7): Linear(in_features=128, out_features=10, bias=True)
)
```

nn.Sequential

```
NeuralNet(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (base_model): Sequential(
    (0): Linear(in_features=3072, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=128, bias=True)
    (5): ReLU()
    (6): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

繼承nn.Module

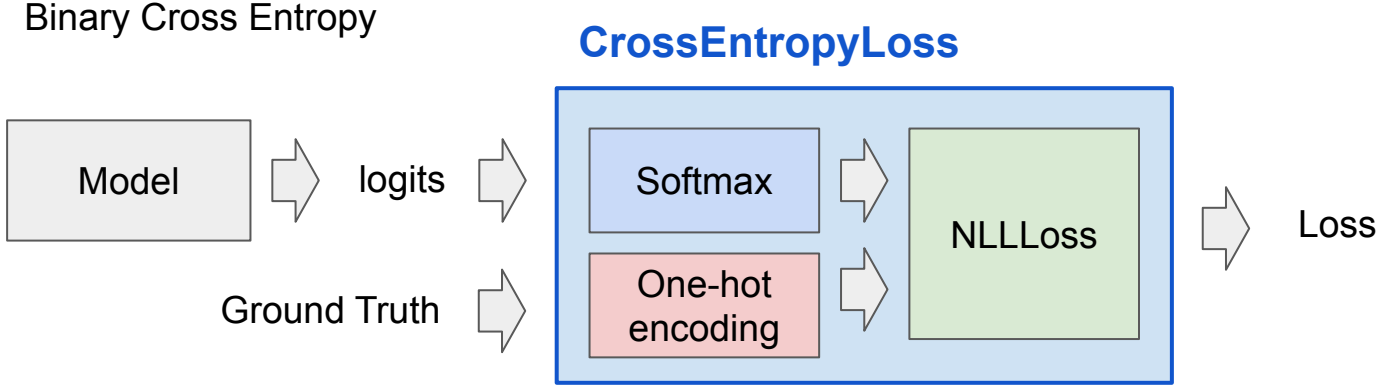
FUSION

torch.nn : Loss function

- 回歸
  - nn.MSELoss: mean squared error
  - nn.L1Loss: mean average error (MAE)
- 分類
  - **nn.CrossEntropyLoss**
    - multi-class classification
    - Softmax + NLLLoss
  - nn.BCELoss
    - binary classification
    - Binary Cross Entropy

PyTorch

```
# 損失函數：計算誤差
loss_fn = nn.CrossEntropyLoss()
pred = model(x) # 預測一批資料
loss = loss_fn(pred, y) # 計算與答案誤差
```



torch.optim

PyTorch

```
# 優化器: 更新模型參數
optimizer = torch.optim.SGD(
    params=model.parameters(), # 要最佳化的模型參數
    lr=1e-1, # learning rate(學習率): 1e-1, 1e-2, 1e-3 ...
)
```

Training per **step**

```
# 模型預測一批資料
pred = model(x)
# 計算損失
loss = loss_fn(pred, y)

optimizer.zero_grad() # 將過去累積梯度歸零
loss.backward() # 透過loss反向傳播計算梯度
optimizer.step() # 更新模型參數
```

$$G = \nabla L(\theta^0)$$

$$\theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

FUSION

Training per **Epoch**

PyTorch

```python
def train_epoch(dataloader, model, loss_fn, optimizer):
    model.train() # 模型轉成訓練模式

    # 依序取出每批資料
    for (x, y) in dataloader:
        x, y = x.to(device), y.to(device) # 資料搬到device

        # 資料丟到模型預測
        pred = model(x)
        # 計算損失
        loss = loss_fn(pred, y)

        optimizer.zero_grad() # 將過去累積梯度歸零
        loss.backward() # 透過loss反向傳播計算梯度
        optimizer.step() # 更新模型參數
```

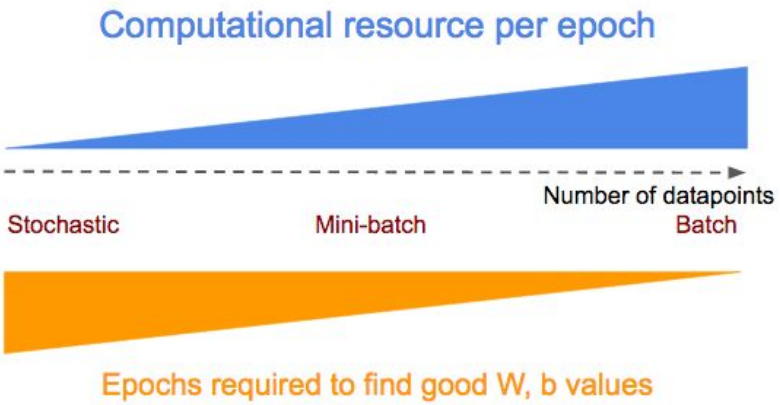
$$G = \nabla L(\theta^0)$$

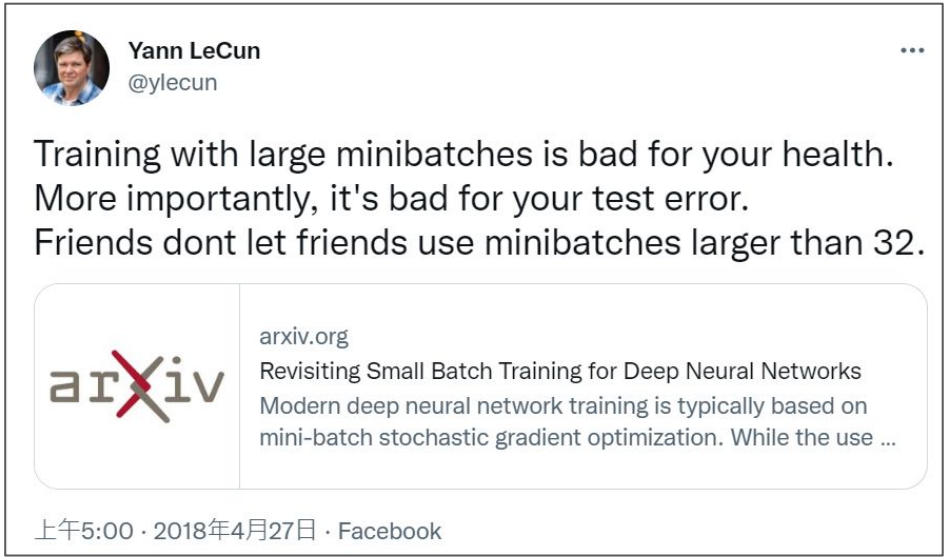
$$\theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

FUSION

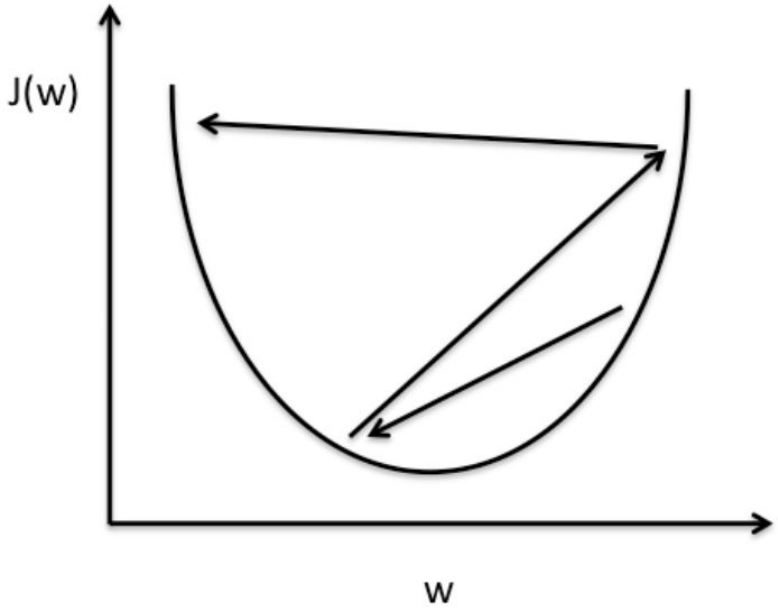
Batch size

- GPU memory
- Larger GPU memory: Larger **batch size**, **model**, **input data**

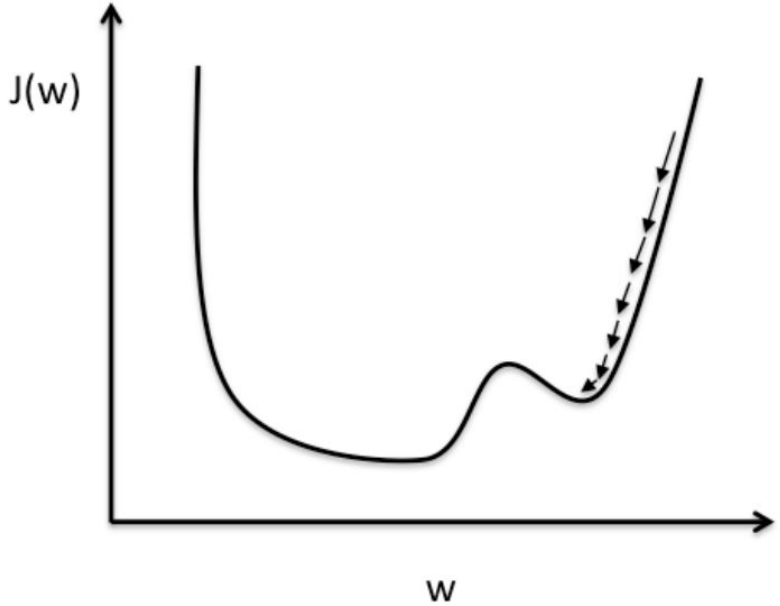




Learning Rate

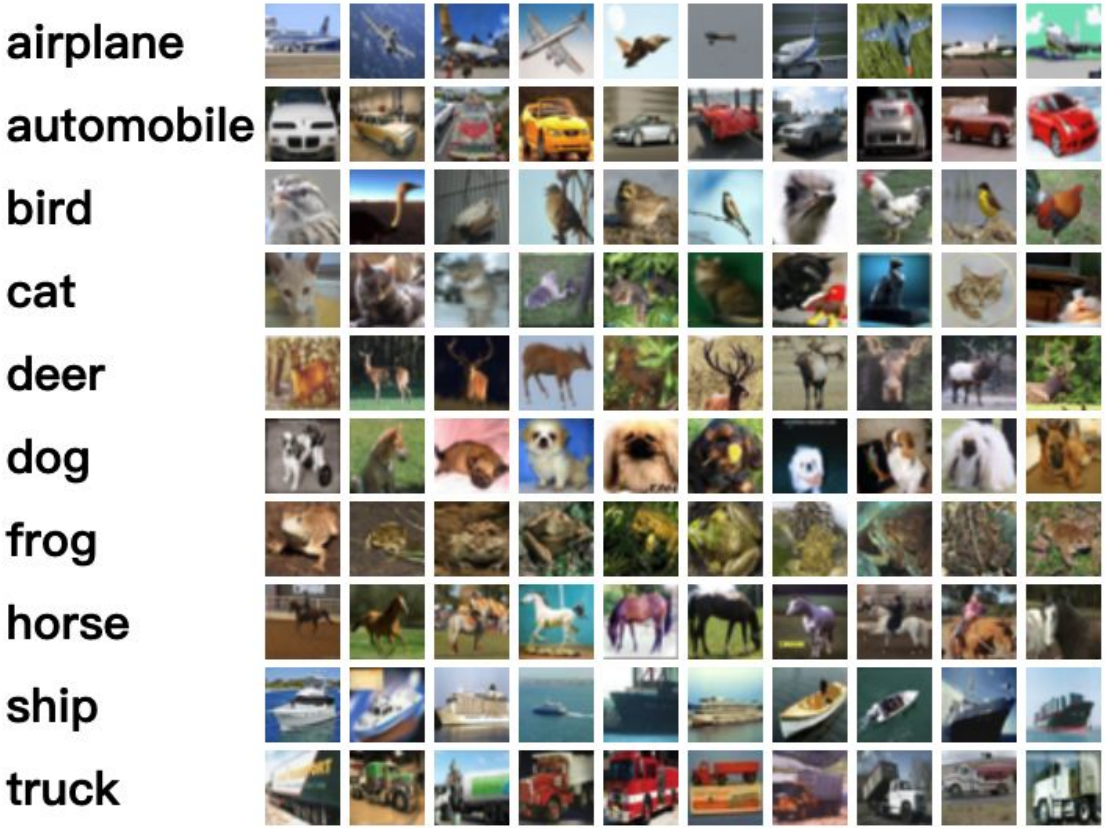


Large learning rate



Small learning rate

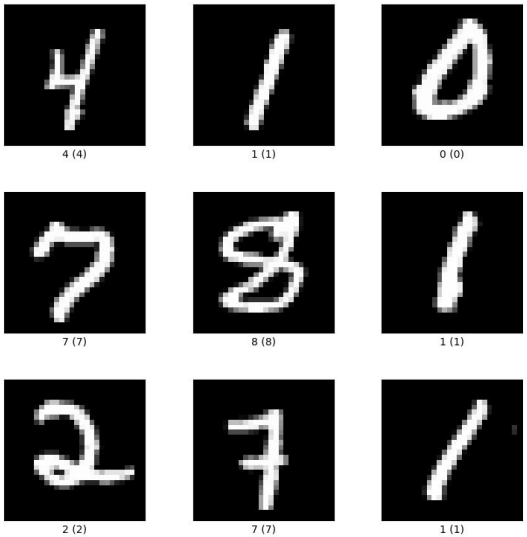
Exercise: CIFAR10



FUSION

HW MNIST

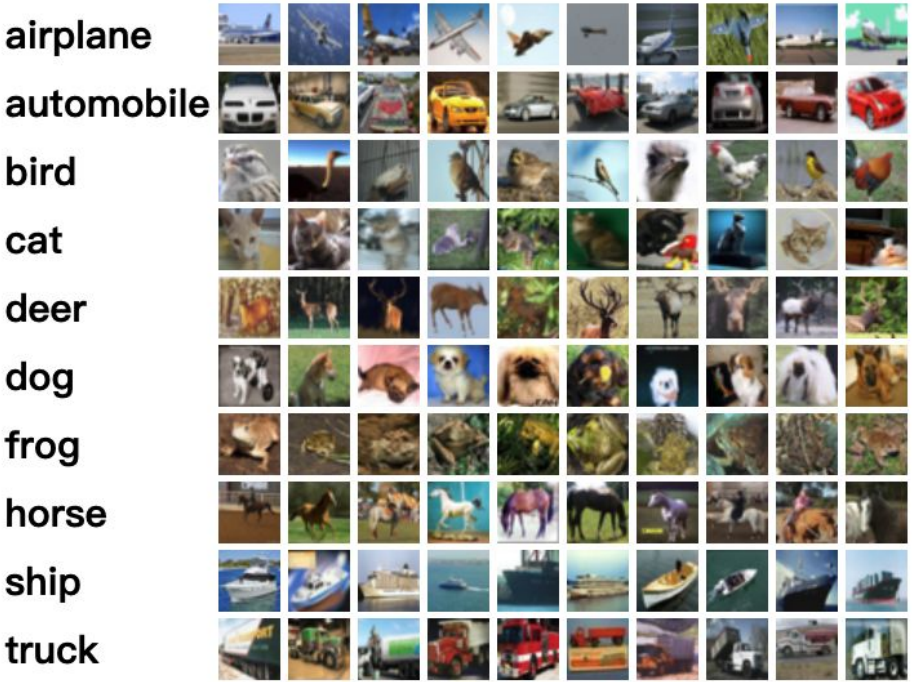
- Kaggle link: https://www.kaggle.com/c/mnist-sai
- Image: (1, 28, 28)



FUSION

HW CIFAR10

- Kaggle link: https://www.kaggle.com/c/sai-cifar10
- Image: (3, 32, 32)



FUSION

Kaggle

- Introduction slides: https://docs.google.com/presentation/d/1scfMHgycgVzqzPHKzaICOWu0F1-kUlqG0Sl0Fj-2SHw

FUSION

Recap

1. Machine Learning
2. Deep Learning
3. Training Process
4. Keras

Model Summary: torchsummary

```python
# 使用 torchsummary 顯示模型架構
import torchsummary

torchsummary.summary(
    model, # 模型
    input_size=(3, 32, 32) # 一筆輸入資料形狀
)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
           Flatten-1                 [-1, 3072]               0
            Linear-2                   [-1, 64]         196,672
              ReLU-3                   [-1, 64]               0
            Linear-4                  [-1, 128]           8,320
              ReLU-5                  [-1, 128]               0
            Linear-6                  [-1, 128]          16,512
              ReLU-7                  [-1, 128]               0
            Linear-8                   [-1, 10]           1,290
================================================================
Total params: 222,794
Trainable params: 222,794
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.03
Params size (MB): 0.85
Estimated Total Size (MB): 0.89
----------------------------------------------------------------
```

FUSION

Model Summary: torchinfo

PyTorch

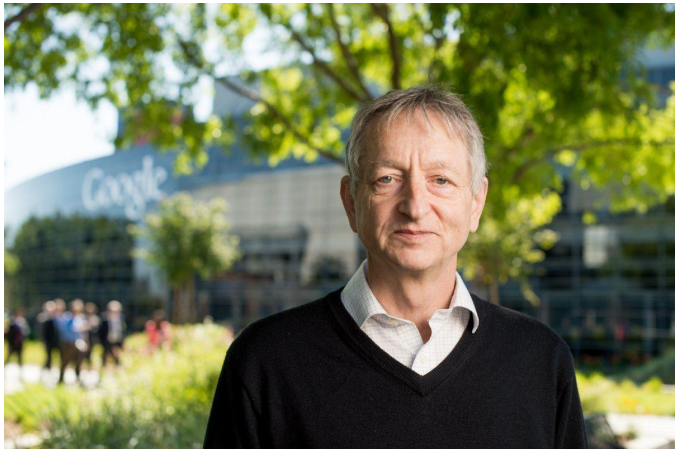
```
# 使用 torchinfo 顯示模型架構
!pip install torchinfo

import torchinfo
torchinfo.summary(
    model,
    input_size=(BATCH_SIZE, 3, 32, 32)
)
```

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
Sequential                               [1024, 10]                --
├─Flatten: 1-1                           [1024, 3072]              --
├─Linear: 1-2                            [1024, 64]                196,672
├─ReLU: 1-3                              [1024, 64]                --
├─Linear: 1-4                            [1024, 128]               8,320
├─ReLU: 1-5                              [1024, 128]               --
├─Linear: 1-6                            [1024, 128]               16,512
├─ReLU: 1-7                              [1024, 128]               --
├─Linear: 1-8                            [1024, 10]                1,290
==========================================================================
Total params: 222,794
Trainable params: 222,794
Non-trainable params: 0
Total mult-adds (M): 228.14
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 2.70
Params size (MB): 0.89
Estimated Total Size (MB): 16.18
==========================================================================
```

FUSION

Gradient Descent 梯度下降

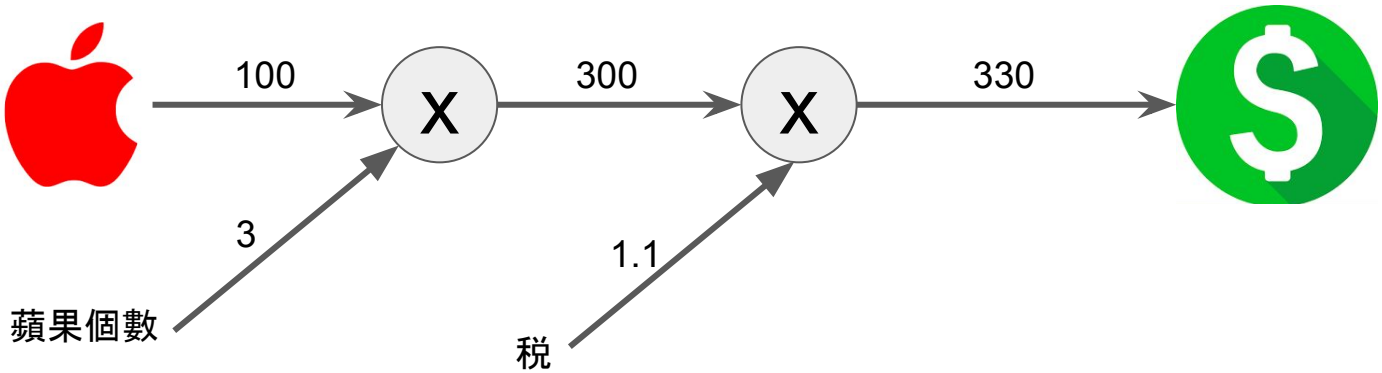
1. 數值微分
2. 反向傳播 : Backpropagation (1986)



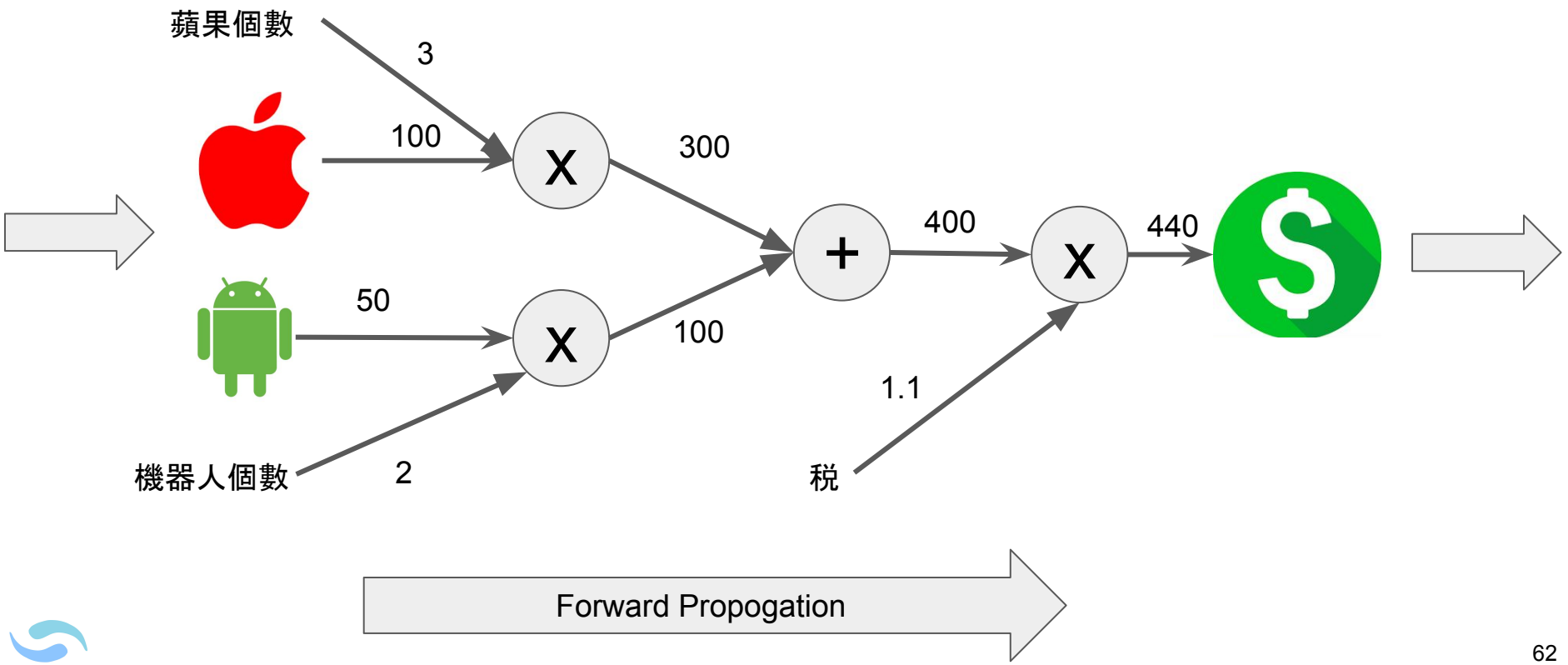
Geoffrey Hinton

計算圖 Computing Graph

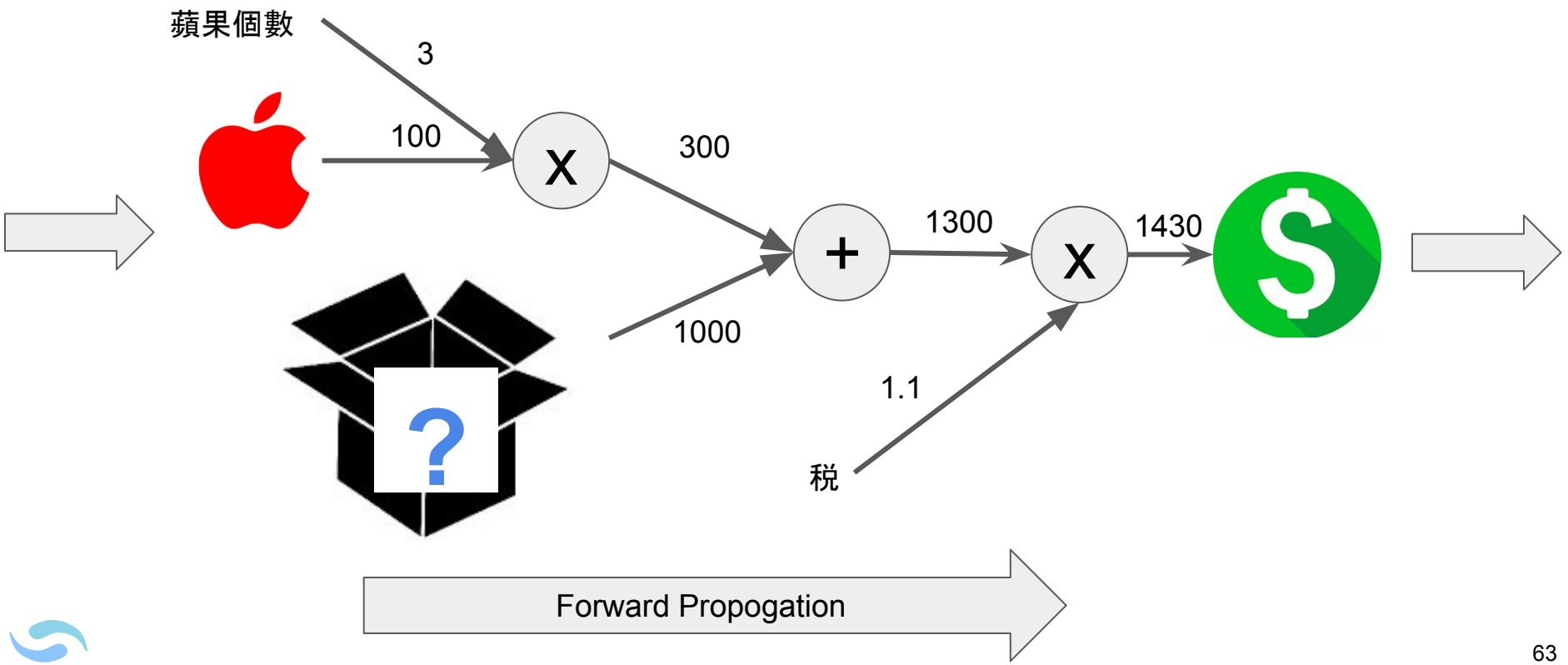
1. 公式
2. 計算圖



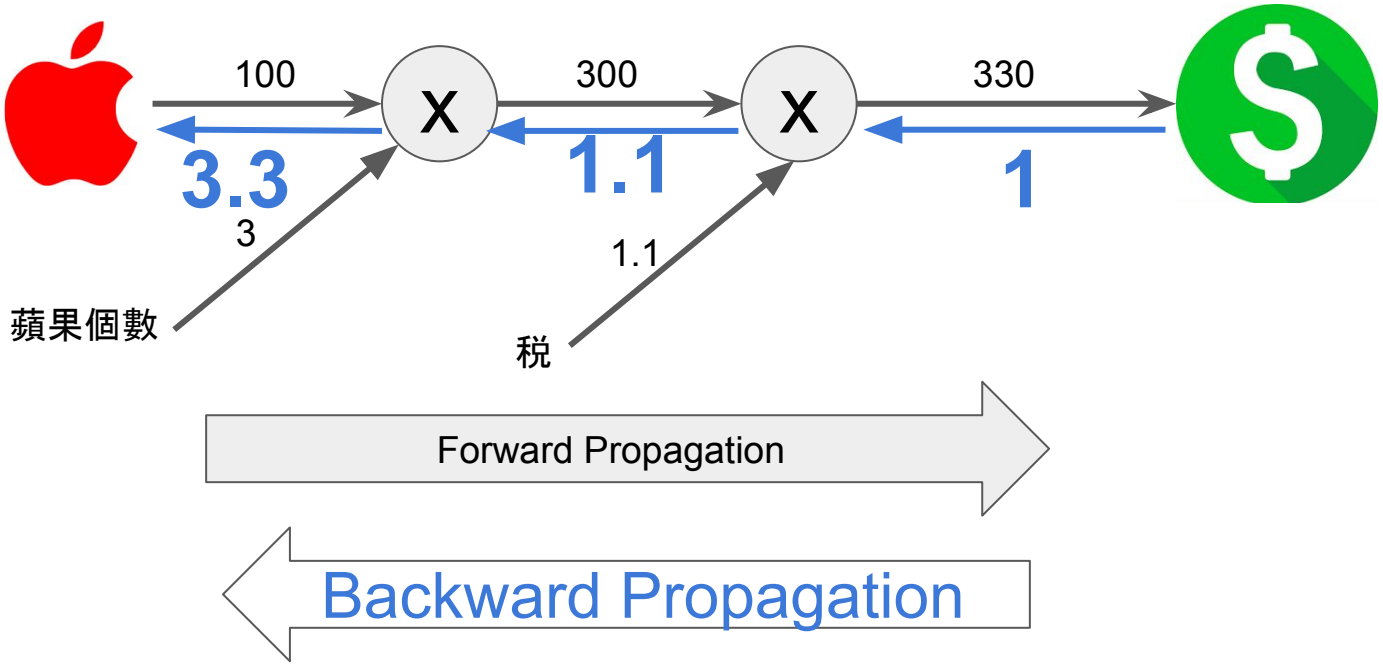
計算圖


蘋果個數
3
100
X
300
+
400
X
440
50
X
100
1.1
機器人個數
2
稅
Forward Propogation

計算圖

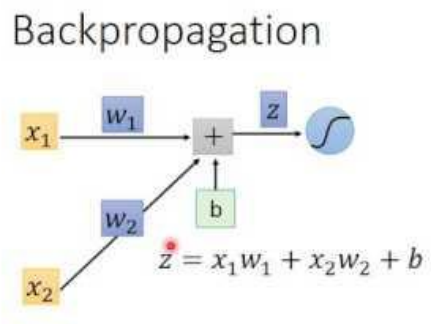


Backpropagation



Resource: Backpropagation

- 台大李宏毅教授
- https://www.youtube.com/watch?v=ibJpTrp5mcE



Error Messages

- **Wrong Tensor shape**

```
nn.Linear(64, 129), # (64)
nn.ReLU(),
nn.Linear(128, NUM_CLASS),
```

```
RuntimeError: Function AddmmBackward returned an invalid gradient at index 1 - got [64, 128] but
expected shape compatible with [64, 129]
```

SEARCH STACK OVERFLOW

0/? [00:00<?, ?it/s]

FUSION

Error Messages

- **GPU out of memory**
  1. Reduce layer **parameters** (output_feature, … etc)
  2. Reduce **batch size**
  3. Reduce **image size**
  4. Buy a **GPU** with larger memory

```
RuntimeError: CUDA out of memory. Tried to allocate 3.73 GiB (GPU 0; 14.76 GiB total capacity; 9.96
GiB already allocated; 1.14 GiB free; 12.56 GiB reserved in total by PyTorch)
```

```
nn.Linear(in_features=IMG_SIZE*IMG_SIZE*3, out_features=100000),
nn.ReLU(),
nn.Linear(100000, 128), # (64) -> (128)
```

```
bs = 10000
train_dataloader = torch.utils.data.DataLoader(train_ds, batch_size=bs, shuffle=True)
val_dataloader = torch.utils.data.DataLoader(val_ds, batch_size=bs)
```

Error Messages

- Wrong devices
  - **x, y, model** should be on the **same device**

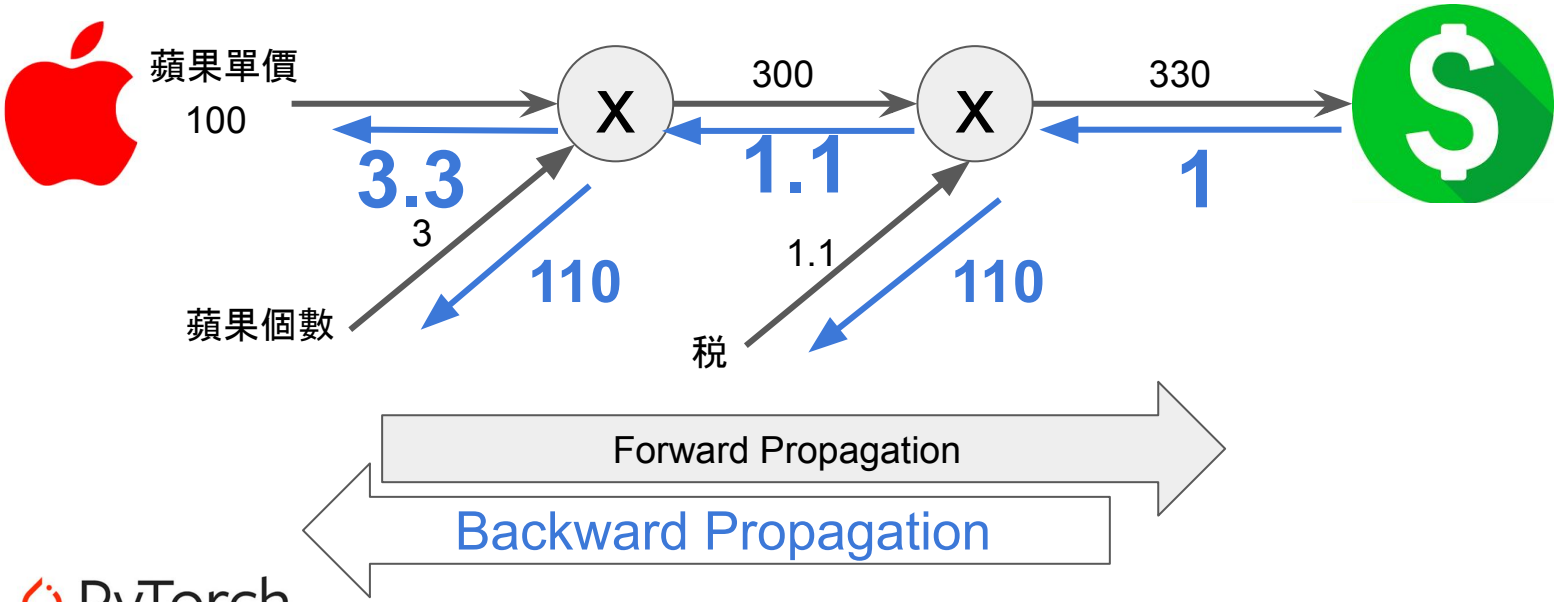
```
RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0
and cpu! (when checking arugment for argument mat1 in method wrapper_addmm)
```

```
model = NeuralNet().to(device)
…
for batch_i, (x, y) in tqdm(enumerate(dataloader), leave=False):
        x, y = x.cpu(), y.cpu()

        pred = model(x)
```

FUSION

Advanced Topic: Backpropagation

- [台大李宏毅教授 Backpropagation]
- Use Backpropagation (BP) to compute the gradient of all parameters



PyTorch

```
loss.backward() # backpropagation to compute gradients
```

Advanced Topic: PyTorch Tensor Operation

- 00_PyTorch_Basic.ipynb
- https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html